# Computational Methods and Graphical Processing Units for Real-time Control of Tomographic Adaptive Optics on Extremely Large Telescopes.

**Sofia Dimoudi**

A Thesis presented for the degree of

Doctor of Philosophy


Centre for Advanced Instrumentation (CfAI)

Department of Physics

University of Durham

England

November 2015

# Computational Methods and Graphical Processing Units for Real-time Control of Tomographic Adaptive Optics on Extremely Large Telescopes

**Sofia Dimoudi**

Submitted for the degree of Doctor of Philosophy

November 2015

## Abstract

Ground based optical telescopes suffer from limited imaging resolution as a result of the effects of atmospheric turbulence on the incoming light. Adaptive optics technology has so far been very successful in correcting these effects, providing nearly diffraction limited images. Extremely Large Telescopes will require more complex Adaptive Optics configurations that introduce the need for new mathematical models and optimal solvers. In addition, the amount of data to be processed in real time is also greatly increased, making the use of conventional computational methods and hardware inefficient, which motivates the study of advanced computational algorithms, and implementations on parallel processors. Graphical Processing Units (GPUs) are massively parallel processors that have so far demonstrated a very high increase in speed compared to CPUs and other devices, and they have a high potential to meet the real-time restrictions of adaptive optics systems. This thesis focuses on the study and evaluation of existing proposed computational algorithms with respect to computational performance, and their implementation on GPUs. Two basic methods, one direct and one iterative are implemented and tested and the results presented provide an evaluation of the basic concept upon which other algorithms are based, and demonstrate the benefits of using GPUs for adaptive optics.

# Declaration

The work in this thesis is based on research carried out at the Centre for Advanced Instrumentation, the Department of Physics, England. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

# Acknowledgements

I would like to thank my supervisors, Richard Myers and Gordon Love for their guidance during my PhD. A special thanks to Richard Myers for his constant support and encouragement, without which, this work may have not been possible.

Also thanks to Alastair Basden for his response and advice to all my queries, and all researchers and staff at the Centre for Advanced Instrumentation who helped me in many ways during my studies.

Finally, I would like to thank my family and friends for standing by me all these years.

This work was funded from a Science and Technology Facilities Council grant.

# Contents

# List of Figures

# List of Tables

# Acronyms

**ALU** Arithmetic Logic Unit

**AO** Adaptive Optics

**API** Application Programming Interface

**AR1** Auto Regressive model of order one

**BCCB** Block-Circulant with Circulant Blocks

**BLAS** Basic Linear Algebra Subroutines

**CCD** Charged Coupled Device

**CG** Conjugate Gradients

**CMOS** Complementary Metal Oxide Semiconductor

**CRS** Compressed Row Storage

**CUDA** Compute Unified Device Architecture

**CuRe** Cumulative Reconstructor

**CuReD** Cumulative Reconstructor with Domain decomposition

**DAC** Digital to Analog Converter

**DARC** Durham AO Real-time Controller

**DASP** Durham AO Simulation Package

**DM** Deformable Mirror

**DMA** Direct Memory Access

**DSP** Digital Signal Processor

**EAGLE** ELT Adaptive Optics for GaLaxy Evolution

**E-ELT** European Extremely Large Telescope

**ELT** Extremely Large Telescope

**EnKF** Ensemble Kalman Filter

**ESO** European Southern Observatory

**ETKF** Ensemble Transform Kalman Filter

**FDPCG** Fourier Domain Preconditioned Conjugate Gradients

**FFT** Fast Fourier Transform

**FPGA** Field Programmable Gate Array

**FPU** Floating Point Unit

**FrIM** Fractal Iterative Method

**FWHM** Full Width at Half Maximum

**GMT** Giant Magellan Telescope

**GPGPU** General-Purpose computation on Graphics Processing Units

**GPU** Graphical Processing Unit

**GS** Guide Star

**HPC** High Performance Computing

**HVA** High Voltage Amplifier

**IDE** Integrated Development Environment

**LAPACK** Linear Algebra PACKage

**LGS** Laser Guide Star

**LLVM** Low Level Virtual Machine

**LQE** Linear Quadratic Estimator

**LQG** Linear Quadratic Gaussian

**MAGMA** Matrix Algebra for GPU and Multicore Architectures

**MAORY** Multi-conjugate Adaptive Optics RelaY

**MCAO** Multi-Conjugate Adaptive Optics

**MGPCG** Multi-Grid Preconditioned Conjugate Gradients

**MOAO** Multi-Object Adaptive Optics

**MPI** Message Passing Interface

**MVM** Matrix Vector Multiplication

**NAOMI** Nasmyth Adaptive Optics for Multi-purpose Instrumentation

**NGS** Natural Guide Star

**OpenCL** Open Computing Language

**PCG** Preconditioned Conjugate Gradients

**PCIe** PCI express

**PSF** Point Spread Function

**PTX** Parallel Thread Execution

**PTX ISA** Parallel Thread Execution Instruction Set Architecture

**RDMA** Remore Direct Memory Access

**RTCP** AO Real Time Control Pipeline

**RTCS** Real Time Control System

**SIMD** Single Instruction, Multiple Data

**SIMT** Single Instruction, Multiple Threads

**SM** Streaming Multiprocessor

**SPD** Symmetric Positive Definite

**SVD** Singular Value Decomposition

**TMT** Thirty Meter Telescope

**TSVD** Truncated Singular Value Decomposition

**UVA** Unified Virtual Address

**VLT** Very Large Telescope

**WFS** Wave Front Sensor

**WHT** William Herschel telescope

# Chapter 1

# Introduction

Currently planned Extremely Large Telescopes (ELTs) are expected to greatly advance astrophysical research, by enhancing observations of exoplanets, measuring properties of the first objects in the Universe, and probing dark matter and dark energy, among others. ELT projects are underway, the biggest of which are the 39m European Extremely Large Telescope (E-ELT) planned by the European Southern Observatory (ESO), the Thirty Meter Telescope (TMT) led by a partnership between the California Institute of Technology, the University of California and national institutions of India, China and Japan, and the 25m Giant Magellan Telescope (GMT) that consists of an international consortium of science institutions. Key properties to enable their scientific goals, are their high sensitivity and high resolution that can be achieved due to their large aperture size. Optical resolution on ground-based telescopes is, however, limited by atmospheric turbulence. Adaptive Optics (AO) systems [1] are currently employed in large ground-based telescopes to compensate for this effect and increase optical resolution by correcting the incoming wavefront and controlling optical components in real-time. Operating within the

real-time limits in adaptive optics is a critical factor in the overall system imaging performance, and is therefore a subject of high priority for research in the design of next generation systems.

Wavefront reconstruction in adaptive optics is typically achieved by solving a system of linear equations. Next generation adaptive optics systems intended for ELTs will employ a very large number of equations, up to the order of $10^5$, bringing a great challenge to the computational requirements of the wavefront reconstruction process. With system configurations such as Multi-Conjugate Adaptive Optics (MCAO) in the plans for future telescopes, systems will be dealing with multiple guide stars, and the phase estimation will be addressed with 3-Dimensional, layered atmospheric turbulence models. Traditional least squares techniques that employ matrix inversion and matrix-vector multiplication may no longer be sufficient in terms of numerical stability. In addition, they will require computational speed and resources that are not available with conventional hardware. While new numerical methods are being studied to reduce the computation cost, a turn towards alternative hardware architectures that can satisfy the long term requirements of the future AO systems is also under consideration.

The estimation of a 3-Dimensional phase perturbation profile, often referred to as atmospheric tomography, is essential to the advancement from the classical AO systems to the next generation wide field of view systems on Very Large Telescopes (VLTs) and ELTs. This step is necessary in order to deal with the problems of a) the limited probability of finding suitable natural reference sources within a narrow field of view (determined by the observation wavelength), and b) the effects that

occur when we use artificial sources to overcome the first problem, which become more severe as the aperture size increases. The scientific goals for the planned ELTs require that there is AO correction for wavelength ranges from 0.6 to 2.4 $\mu$m, and for fields of view from 1 arc second to 5 arc minutes. Because of this, artificial reference sources are needed, and they need to be multiple in order to provide the desired field of view and increase the chances of finding natural reference sources, provide correction at shorter wavelengths, and compensate for the effect of the limited range of the sources. Where higher imaging resolution is required, then correction must also be applied over multiple layers of atmospheric turbulence. However, atmospheric tomography is both much more ill-posed than the model currently employed in AO, which compromises accuracy and amplifies noise, and also much more data intensive, which affects computational speed. Both of these issues have an impact on the imaging performance of the system. To cope with the solution accuracy, it is necessary to use an optimal solver, that will make use of prior statistics of the turbulence, to counterweight poorly sensed optical turbulence components against system noise. The Minimum Variance Estimator [2] is found to be the optimal solver, as it optimises the solution by minimising the residual wavefront variance, which is also the final objective in adaptive optics. It should be noted that this applies only to the static solution. If temporal correlations of the optical signal are used, then the optimal solution is given by a predictive controller, such as the Linear Quadratic Gaussian (LQG) [3], although it is much more computationally demanding. The problem of the real-time temporal response is being addressed with the study of numerical algorithms that apply to the chosen solvers and aim at making computations faster

while keeping an acceptable level of solution accuracy. The general objective is to sparsify the computational structures in an appropriate manner, using a model that correctly represents the AO problem, and applying to this mathematical techniques that result in fewer computations, less memory storage and transfers. Even with the use of such an advanced method, the real-time goals for atmospheric tomography reconstruction are not achievable with current CPU technology, therefore there is a need for different hardware platforms to be considered. On the other hand, if a hardware accelerator can provide enough computing power to apply the solution using simple matrix vector multiplications on a minimum variance reconstructor, then this is also a desirable option that allows using available software libraries and avoids building and maintaining complex custom programs.

This thesis assesses the use of Graphical Processing Units (GPUs) as hardware accelerators for real time wavefront reconstruction with atmospheric tomography in conjunction with computational methods that have been proposed in recent years.

## 1.1 Synopsis

The remainder of this document is organised as follows:

Chapter 2 covers the basic theoretical background on atmospheric turbulence and adaptive optics, as well as an introduction to the next generation adaptive optics systems configurations and atmospheric tomography. Chapter 3 provides a review of the most notable advanced computational algorithms that have been proposed for ELT and atmospheric tomography, and discusses issues concerning their parallel implementation. In chapter 4 I describe the GPU architecture and programming

model, and the current provisions of this technology to developers, and discuss how these devices could fulfil the basic latency requirements in an AO system. In chapter 5 I present the method of mapping a direct and an iterative algorithm onto the GPU, and chapter 6 shows experimental results and discusses the findings. Finally, in chapter 7, I draw my conclusions and suggestions for future work in the subject.

# Chapter 2

# Adaptive Optics and Real-time Control

In this chapter, I present the basic background theory on atmospheric turbulence and the principles of an adaptive optics system, and introduce the concept of real-time control. I then describe the problem of atmospheric tomography and the next generation adaptive optics configurations on which real-time control will be applied.

## 2.1 Background theory

Adaptive optics systems aim at correcting the effects of atmospheric turbulence. A basic understanding of the characteristics of this process is required in order to design parts of such systems. A brief presentation of the main concepts that describe the effects of atmospheric turbulence on imaging follows.

### 2.1.1 Atmospheric Turbulence

The atmosphere is characterized by a turbulent flow of air masses of varying sizes, caused by temperature variations that occur as solar heating dissipates through the atmospheric medium and gives rise to random changes in local wind velocity. These temperature variations also result in local changes in density that affect the refractive index of the air, which in turn causes optical beam paths to divert. The strength of the turbulence is measured by the refractive index structure constant $C_n^2$ which is not constant, but has a profile that depends on altitude, location and time, including seasons, and can be used to evaluate the atmospheric conditions at particular locations that are candidates for astronomical sites. The $C_n^2$ variation as a function of altitude is an important representation as it allows one to obtain a measure of the total optical degradation along a vertical propagation path. A widely used model to express this relationship is the Hafnagel-Valey (H-V) model resulting from studies in [4–6]:

$$C_n^2(h) = 5.94 \times 10^{-23} h^{10} \left(\frac{W}{27}\right) e^{-h} + 2.7 \times 10^{-16} e^{-2h/3} + A e^{-10h}, \qquad (2.1.1)$$

where $h$ is the altitude in kilometres and $C_n^2$ is in units of $\mathrm{m}^{-2/3}$. $A$ and $W$ are adjustable parameters to fit the site conditions, corresponding to the surface turbulence strength and upper layer wind velocity respectively. Figure 2.1 has an illustration of a measurement-based $C_n^2$ turbulence profile.

Atmospheric turbulence has a random nature and can be quantitatively described using statistical analysis. Kolmogorov [8] studied a model for the velocities in a fluid medium, which analysed the structural properties of turbulence, and formed

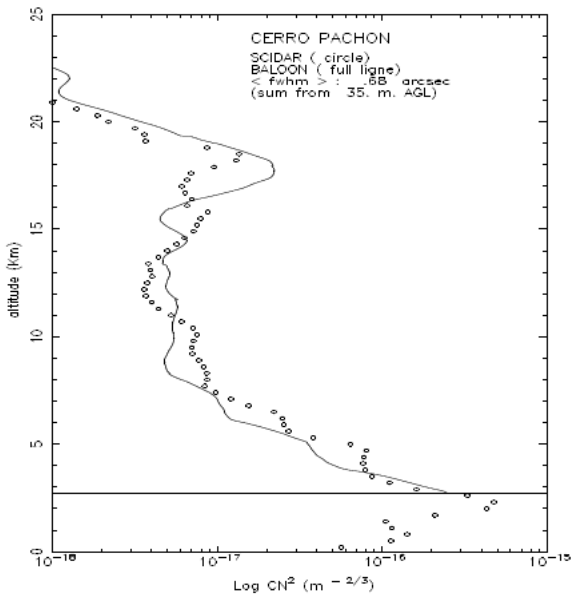



Figure 2.1: $C_n^2$ profile measured for the 1998 Gemini site testing survey at Cerro Pachon (Vernin et al. [7]). Comparison of G-SCIDAR (circles) and Balloon (solid line) measurements over a whole year period.

the basis for much of today's theoretical and practical applications in the field of fluid mechanics, and is also commonly used in AO design. His model adopts the eddy-viscosity concept for turbulent flow, where the term 'eddy' broadly refers to a localized turbulent motion. According to this model, the solar energy is passed to the medium as large scale motion and cascades to smaller and smaller scales in the form of kinetic energy, until the viscosity of the fluid can effectively dissipate it. It defines an inner and an outer spatial scale, $l_0$ and $L_0$, within which the energy is transmitted, and at the smaller scale, it is dissipated as heat. A schematic of this process is shown in figure 2.2..

The Kolmogorov model assumes that turbulence is locally isotropic and has self-similarity at scales much smaller than the outer scale, that is, the small scale motions behave the same in all directions, and their statistics have a universal form that is independent of the larger scales. Based on these principles, the refractive index

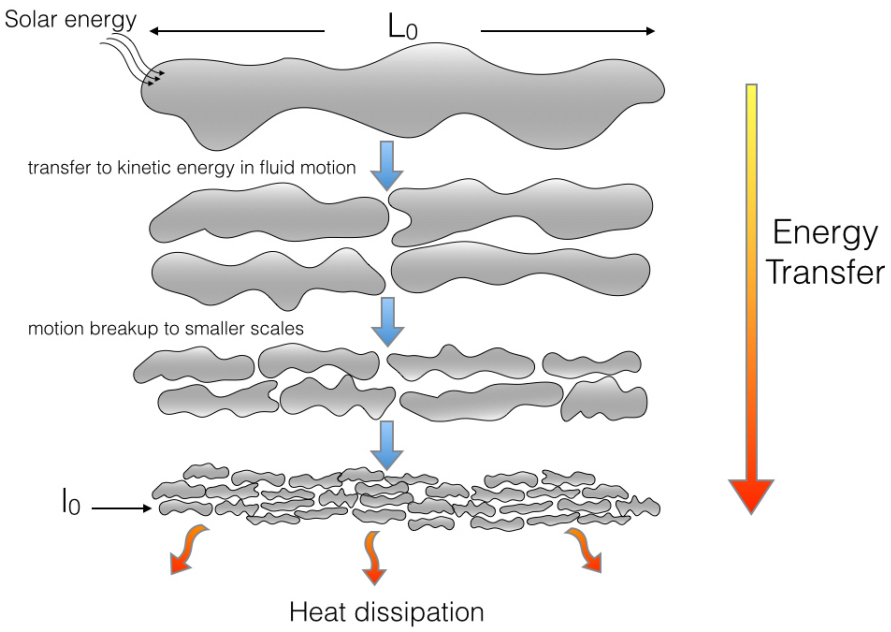



Figure 2.2: Kinetic energy flow in the atmosphere through outer to inner spatial scales.

variations can be described by the structure function:

$$D_n(r) = \left\langle [n(\rho - r) - n(\rho)]^2 \right\rangle \tag{2.1.2}$$

where $r$ is the separation between two points in space starting from point $\rho$, $n(\rho)$ denotes the refractive index at the specified point $\rho$, and the angle brackets denote ensemble average. For turbulence within the Kolmogorov inner and outer scale (the inertial subrange), 2.1.2 becomes:

$$D_n(r) = C_n^2 r^{2/3}, \qquad l_0 < r < L_0 \tag{2.1.3}$$

where $C_n^2$ is the refractive index structure constant described earlier.

The Kolmogorov refractive index power spectrum in three dimensions is ex-

pressed by its power spectral density after Tatarskii [9]:

$$\Phi_{\mathrm{n}}(k) = 0.033 C_n^2 k^{-11/3} \tag{2.1.4}$$

where $k = 2\pi/r$ is the three-dimensional spatial wave number. Expression 2.1.4 is valid only within the inertial range between $l_0$ and $L_0$, but the size of the outer scale varies from a few metres to over 100 metres, which can be comparable to the size of a modern telescope. Therefore, in determining the optical effects, the Kolmogorov spectrum does not account for effects that may occur due to the size of the outer scale being similar to the telescope aperture size. For these purposes, the von Karman [10] spectrum can be used, which relates the outer scale size to the power spectrum. A modified version that accounts for both inner and outer scale is the following [11]:

$$\Psi(k) = 0.033 C_n^2 (k^2 + k_0^2)^{-11/6} e^{-k^2/k_i^2} \tag{2.1.5}$$

where $k_i = 5.91/l_0$ and $k_0 = 2\pi/L_0$ relate to the inner and outer scale of turbulence, and 2.1.5 is valid in the inertial subrange with $2\pi/L_0 \leq k \leq 2\pi/l_0$.

### 2.1.2 Effects on Image Structure

A wavefront is a surface over which an optical wave has a constant phase. As light travels from distant light sources it is transmitted in flat, or plane waves. Variations in the refractive index that occur in the turbulent atmosphere cause local phase delays across the wavefront according to the size of the turbulent cells, that cause the flat surface to be deformed. As waves travel along vertical light beam paths of

height $z$, the phase of the wavefront produced at the telescope aperture at point $x$ due to the refractive index fluctuation $n(z)$ is

$$\phi(x) = k \int n(x, z)\, dz \tag{2.1.6}$$

with the wavenumber $k = 2\pi/\lambda$, where $\lambda$ is the wavelength of the light. The variance of the phase over two points on a two-dimensional wavefront surface is given by the phase structure function

$$D_\phi(r) = \left\langle [\phi(x) - \phi(x - r)]^2 \right\rangle \tag{2.1.7}$$

where $r$ is the distance between the two phase points in two dimensions. At the aperture, as the surface is integrated along the propagation path, the phase structure function can be expressed in terms of the refractive index structure function as

$$D_\phi(r) = 2.91k^2(\cos\gamma)^{-1} \int C_n^2(h)\, dh\, r^{5/3} = 6.88 \left(\frac{r}{r_0}\right)^{5/3} \tag{2.1.8}$$

where $\gamma$ is the zenith angle. The term $r_0$ is called the Fried parameter after Fried [12], which is a measure of the strength of turbulence and is given by:

$$r_0 = \left[0.423k^2(\cos\gamma)^{-1} \int C_n^2(h)\, dh\right]^{-3/5} \tag{2.1.9}$$

As it can be seen in 2.1.9, $r_0$ varies with wavelength, thus its value is defined for particular wavelengths. The Fried parameter is also called the turbulence coherence length, as the root mean square phase difference over a circular area of diameter $r_0$

is 1 radian. This results from the following relationship that defines the rms phase over a circular area of diameter $d$

$$\sigma_\phi^2 = 1.03 \left(\frac{d}{r_0}\right)^{5/3} \tag{2.1.10}$$

It should be noted that the above expression can be used in its general form to obtain the mean-square residual variance, or fitting error of a compensated wavefront [13]:

$$\sigma_F^2 = \alpha_F \left(\frac{d}{r_0}\right)^{5/3} \tag{2.1.11}$$

where the coefficient $\alpha_F$ depends on the amount and type of correction applied.

The coherence length is an important term because it can be used to describe many optical effects of turbulence with a single parameter. For the image resolution of a telescope, $r_0$ can be used to express the angular limit to resolution due to atmospheric seeing:

$$\theta_s = \frac{\lambda}{r_0} \tag{2.1.12}$$

The turbulence spectrum in the spatial domain can also be expressed in terms of $r_0$:

$$\Phi(\xi) = \frac{0.023}{r_0^{5/3}}\, \xi^{-11/3}, \tag{2.1.13}$$

where $\xi$ is in $\mathrm{m}^{-1}$. This spectrum contains the spatial frequencies over which there are phase variations produced by turbulence, and can be used to describe the resulting optical aberrations. The integration of the spectrum over the spatial frequencies on the measured area results in an expression of the overall wavefront variance of

that area:

$$\sigma^2 = \int \Phi(\xi)\,\mathrm{d}^2\xi \tag{2.1.14}$$

Temporal characteristics of atmospheric turbulence are also of importance to its effect on image formation. A structure function can be defined to express temporal variations of phase in space in analogy to 2.1.7:

$$D_\phi(\tau) = \left\langle [\phi(x,t) - \phi(x,t+\tau)]^2 \right\rangle = 6.88 \left(\frac{\bar{\upsilon}\tau}{r_0}\right)^{5/3} \tag{2.1.15}$$

where $\bar{\upsilon}$ is taken to be the mean wavefront phase propagation velocity, which is an approximation that neglects the temporal variations caused by each turbulent layer, as they do not have a significant effect when it comes to AO correction [14]. The time within which the rms wavefront difference is less than 1 radian, or coherence time is then found:

$$\tau_0 = 0.314\frac{r_0}{\bar{\upsilon}} \tag{2.1.16}$$

which is important in determining the required rate of correction using adaptive optics.

### 2.1.3 Wavefront aberrations

The optical wavefront can be decomposed to a series of orthogonal basis two dimensional functions, defined over a unit circle. A polynomial expansion, which was introduced by Zernike [15] and further analysed by Noll [16], is commonly used in optical physics, due to their simple analytical form among other things. This is also termed a modal wavefront representation. In polar coordinates $r$ and $\theta$, the Zernike

modes for a circular aperture have the form

$$Z_j(r,\theta) = \sqrt{n+1}R_n^m \begin{cases} \sqrt{2}\cos(m\theta) & j \text{ is even} \\ \sqrt{2}\sin(m\theta) & j \text{ is odd} \\ 1 & m = 0 \end{cases} \tag{2.1.17}$$

with

$$R_n^m = \sum_{s=0}^{(n-m)/2} \frac{(-1)^s (n-s)!}{s![(n+m)/2-s]![n-m)/2-s]!}\, r^{n-2s}. \tag{2.1.18}$$

$R_n^m$ are the Zernike radial polynomials, $n$ is the radial degree, and $m$ the azimuthal frequency.

Wavefront distortions over a circular aperture are expressed as a sum of Zernike polynomials:

$$\phi(\mathbf{r}) = \sum_j \alpha_j Z_j(\mathbf{r}) \tag{2.1.19}$$

where $\mathbf{r} = \mathbf{r}(r,\theta)$ is the polar vector, and $\alpha_j$ are the Zernike coeffcients

$$\alpha_j = \int \phi(\mathbf{r}) Z_j(\mathbf{r}) W(\mathbf{r})\, \mathrm{d}\mathbf{r} \tag{2.1.20}$$

for a weighting function $W(\mathbf{r})$ corresponding to a telescope pupil. For $j = 1$ we get the average phase over the aperture, which is also referred to as overall piston. The piston-removed wavefront variance in terms of the Zernike expansion becomes

$$\sigma_\phi^2 = \sum_{j=2}^{\infty} \langle \alpha_j^2 \rangle. \tag{2.1.21}$$

If an adaptive optics system is able to correct the aberrations that amount to $N$

Zernike modes, then the residual phase variance will be

$$\sigma^2_{\phi_N} = \sigma^2_{\phi} - \sum_{j=2}^{N} \langle \alpha_j^2 \rangle. \tag{2.1.22}$$

Zernike polynomials correspond to particular wavefront shapes caused by aberrations. Table 2.1 lists the first 8 Zernike modes with their corresponding aberration and figure 2.3 illustrates their shapes after the piston mode.

The residual phase variance related to Zernike modes may also be expressed in terms of $r_0$ as shown by Noll [16]:

$$\sigma^2_{\phi_N} = A_N \left( \frac{D}{r_0} \right)^{5/3}. \tag{2.1.23}$$

In the above expression, $D$ represents the telescope aperture diameter. The first 10 values of the parameter $A_N$ are calculated by Fried [12] and Noll [16], and the latter also provides an approximation of $A_N$ for $N > 10$.

| **j** | **n** | **m** | **Zernike Polynomial** | **Aberration** |
|---|---|---|---|---|
| 1 | 0 | 0 | $1$ | Piston |
| 2 | 1 | 1 | $2r\cos\theta$ | Tilt |
| 3 | 1 | -1 | $2r\sin\theta$ | Tilt |
| 4 | 2 | 0 | $\sqrt{3}(2r^2-1)$ | Defocus |
| 5 | 2 | -2 | $\sqrt{6}r^2\sin 2\theta$ | Astigmatism |
| 6 | 2 | 2 | $\sqrt{6}r^2\cos 2\theta$ | Astigmatism |
| 7 | 3 | -1 | $\sqrt{8}(3r^2-2r)\sin\theta$ | Coma |
| 8 | 3 | 1 | $\sqrt{8}(3r^2-2r)\cos\theta$ | Coma |

Table 2.1: First 8 Zernike polynomials with corresponding optical aberrations they represent.

Zernike polynomials are useful for representing optical aberrations for the design and analysis of optical systems. In adaptive optics, they are used to measure the

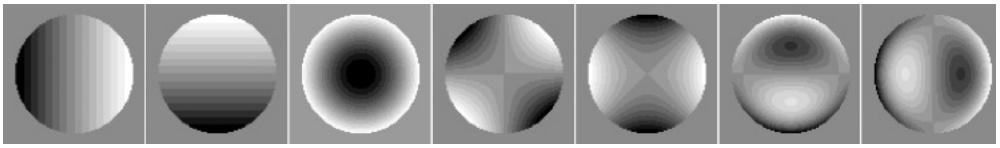


Figure 2.3: Shapes of single Zernike polynomials on a circular area.

imaging performance after compensation, but also as a basis for correction. When an AO system uses Zernike polynomials to reconstruct the wavefront, it is said to be using a modal reconstructor. A modal reconstructor works by either directly sensing the wavefront modes, or translating wavefront measurements to a modal representation (depending on the sensing method used), and then use this representation to create a linear system of equations. The solution of that system provides, in the case that the Zernike expansion is employed, the set of Zernike coefficients, which are then used in a further step to construct and fit the shape of the wavefront to the correcting device. Modal correction is usually preferred for reconstruction of low order aberration modes.

### 2.1.4 Image performance

The optical system response is determined by the Point Spread Function (PSF) of the system, which describes the distribution of the light intensity on the image plane as a response of the optics to a point source. The diffraction-limited angular resolution of a telescope of diameter $D$ at incoming light wavelength $\lambda$ is defined by the Rayleigh criterion, which states that two diffraction-limited PSFs are resolved if their separation is at least $2\pi\lambda/D$, where the first diffraction minimum of one image coincides with the maximum of the other. Considering that the image of a

point source has a nearly gaussian intensity profile, the image size is given by the Full Width at Half Maximum (FWHM) of the PSF, which is the angular distance between two points where the intensity reaches half of its maximum value, and is approximated by $\lambda/D$. In the presence of atmospheric turbulence, this is also referred to as 'seeing' disc, and its size depends on the turbulence coherence length, and defined by $\lambda/r_0$. The FWHM is an acceptable criterion for image quality, provided that the image profile does not change significantly with seeing conditions, apart from its width, which is the case for turbulence affected images [14]. These uncompensated images (figure 2.4, centre) have a nearly Gaussian PSF. Images that result from AO correction however, are characterised by a narrow core and a surrounding halo (figure 2.4, right). Depending on the amount of compensation, the amount of light in the halo due to the residual wavefronts varies. The FWHM measure in this case may not represent correctly the image quality of the AO compensated image, if the halo is higher than the half maximum, where the FWHM is measured. To describe image performance in AO compensated images, the most useful measure is the Strehl ratio, which is more sensitive to residual wavefront errors. The Strehl ratio $SR$ is the ratio of the peak intensity of the corrected image $I_{im}$, to that of a diffraction-limited image $I_{diff}$:

$$SR = \left( \frac{I_{im}}{I_{diff}} \right) \approx e^{-\sigma^2} \tag{2.1.24}$$

The last approximation in (2.1.24) is valid when the residual wavefront rms error $\sigma$ is less than 2 radians.

Another useful measure is the Encircled (or Ensquared) Energy (EE), which ex-

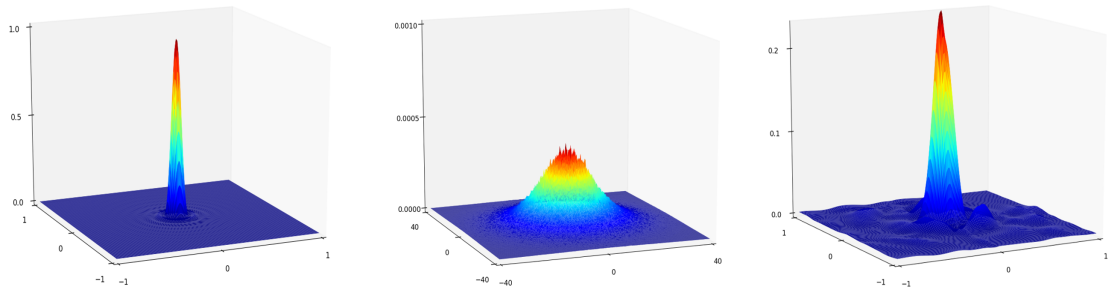


Figure 2.4: Illustration of the PSF. Left: theoretical PSF. Centre: uncompensated PSF. Right: partially corrected PSF with SR of 22% .

presses the concentration of energy within a given circular (or square pixel diameter) of the image. This is very useful for applications of high resolution spectroscopy.

### 2.1.5 Adaptive Optics System Components

An adaptive optics system consists of three main parts: a Wave Front Sensor (WFS), a wavefront corrector, and a control computer, that work in sequence to provide an optical correction of the distorted incoming wavefront (figure 2.5). The wavefront sensor is a sensing device that collects incoming light from a reference source, called a Guide Star (GS) and, in its most common form, measures the local slopes across the aperture. The correcting device is a Deformable Mirror (DM) which is an electrical-driven device that has the ability to change its mirror shape. The control computer computes the mirror commands required to reconstruct the wavefront, to be applied to the DM.

The most common type of wavefront sensor is the Shack-Hartmann WFS (figure 2.6). It consists of a lenslet array that is positioned at a conjugate pupil plane on the telescope and focuses light to an image detector, usually a Charged Coupled Device (CCD) camera. If the wavefront of the incoming light is planar, a point

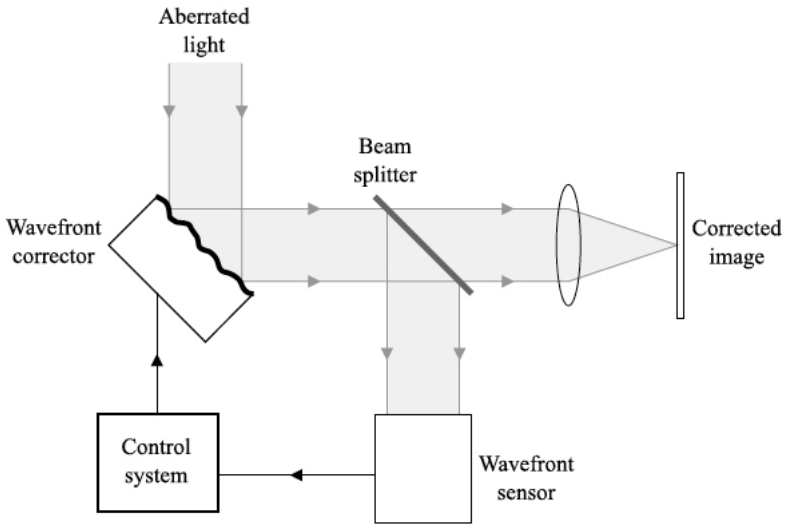



Figure 2.5: Basic parts of an adaptive optics system

source image will be focused at the centre of each CCD sub-aperture. When the wavefront is deformed, a local deformation will divert the focus from a lenslet to an arbitrary position on the corresponding sub-aperture, depending on the angle of arrival of the incoming beam. The centre of gravity of the local spots on the sub-apertures can then be used to reconstruct the local disturbances.

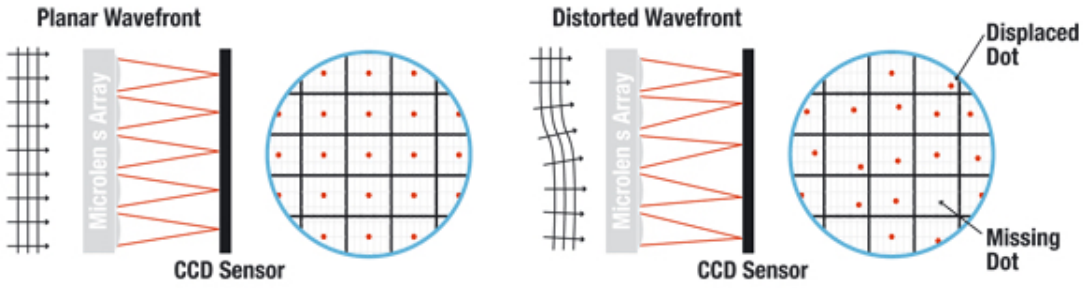



Figure 2.6: Shack-Hartmann wavefront sensor principle (Thorlabs Inc).

Wavefront correctors are used to produce a controlled reflection that will optically remove the induced aberrations. For the low-order aberrations (overall image motion), most systems use a fast steering mirror, which can be tilted in two directions to restore the total angle of arrival of the image. The deformable mirror

handles the higher order deformations by providing reflections corresponding to image sub-apertures. The most used types are segmented mirrors and continuous facesheet mirrors (figure 2.7).

A segmented mirror consists of many small flat mirror segments, each mounted on an actuator, a motorized support able to provide movement in three degrees of freedom (piston, tip and tilt). They have a large dynamic range on their stroke, a good frequency response, and easy maintenance, but they suffer from edge effects due to the gaps between the segments, which are however only important for very low-contrast applications. An example of an AO system using a segmented DM is the Nasmyth Adaptive Optics for Multi-purpose Instrumentation (NAOMI) system on the 4.2 m William Herschel telescope (WHT) [17], which has 76 segments.

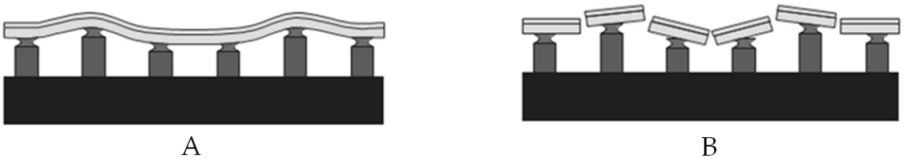



Figure 2.7: (A) Continuous facesheet discrete actuator DM. (B) Segmented DM.

Continuous facesheet mirrors use a thin faceplate of reflective material supported by piston-only actuators that can be one of several forms [13, 14, 18]. The most widely used type is DMs which employ an array of discrete, piezo-electric actuators under the facesheet, mounted on a stable baseplate. A single piezo-electric element has a very limited yield in movement compared to the requirement in AO systems, so to increase the stroke, actuators are manufactured with stacked piezo-electric elements that provide a linear increase in movement with the same voltage applied.

The absence of surface gaps between actuators and the fact that less actuators are required, is an advantage compared to segmented mirrors, but one drawback is the actuators mechanical coupling, that requires additional processing for fitting the correct shape to the mirror. Continuous discrete actuator mirrors are used, for example in the Keck Observatory [19] and the VLT AO system [20].

Figures of merit for the choice of such DMs are their dynamic range, the required voltage, the total size and inter-actuator spacing, the temporal response and the number of actuators. Also of importance, particularly for systems without a DM feedback loop, is hysteresis, which expresses a non-linearity between voltage and displacement, affecting the actuation precision and consequently the control bandwidth and the residual wavefront error. Hysteresis is material dependent and can be at the order of 10% for piezoelectric DMs. A recent comparison of DM technologies for 8m and 40m class telecopes AO can be found in [21].

The function of the control computer is to reconstruct the wavefront from the available measurements, and to fit the result to the DM.

## 2.2 Adaptive Optics control

Adaptive optics control is implemented as a servo control loop, either with feedback, which is the most common, or as feed-forward systems [13, 14, 22]. The function of the control loop is to optimise the output by minimising the residual phase. This process consists of a static and a dynamic part. The static part is the computation of the mirror voltages by performing a reconstruction of the phase from the collected measurements at a given time step, and the dynamic part is to ensure the stability

of the feedback loop, taking into account also the temporal characteristics of the turbulence.

### 2.2.1 Control Loop

The basic model for an AO control loop is shown in the block diagram of figure 2.8. This conventional form of an AO system operates in a closed feedback loop [18]. At

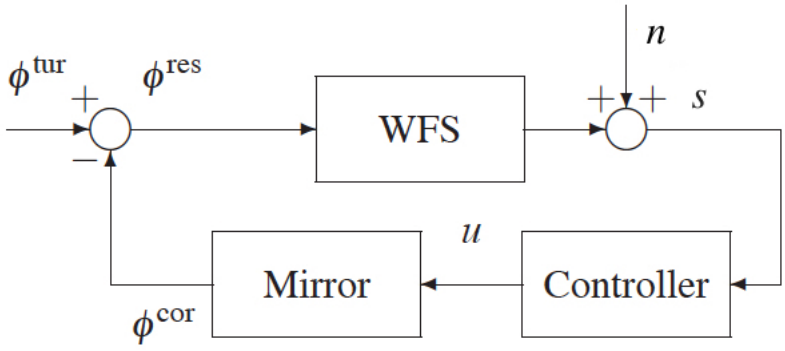



Figure 2.8: Adaptive optics control diagram

time $t$, the system has as input the phase $\phi_{tur}$ of the wavefront affected by atmospheric turbulence, which is measured by the wavefront sensor. The measurements $s$ with additive noise $n$ are processed by the controller that estimates the mirror commands $u$, and feedback $\phi_{cor}$ applied after a wavefront correction stage by the deformable mirror. The output is then the residual wavefront phase $\phi_{res}$. The input - output relationship in discrete time steps $k$ for this system can be written:

$$\phi_{res}(k) = \phi_{tur}(k) - \phi_{cor}(k). \tag{2.2.25}$$

The corrected phase is derived from the mirror commands in the previous step:

$$\phi_{cor}(k) = Nu(k-1), \tag{2.2.26}$$

where $N$ is an influence matrix that fits the control voltages to the mirror. The measurements at time step $k$ are produced from the incoming phases at time $k-1$:

$$s(k) = \Gamma\phi_{tur}(k-1) + n, \tag{2.2.27}$$

where $\Gamma$ is an operator that maps the phase to the measurement device.

In order to control the optical wavefront, that is, to minimise $\phi_{res}$, the previous states are incorporated to the current state using an integrator, such that the mirror commands at time step $k$ are given by:

$$u(k) = -Rs(k) + u(k-1), \tag{2.2.28}$$

where R is the reconstruction operator that calculates phase estimates from measurements.

### 2.2.2 Wavefront reconstruction

The central part of the control loop is to reconstruct the incoming wavefront from a set of discrete sensor measurements by estimating the best fit to these measurements to be applied to the correcting device. In this sense, wavefront reconstruction can be classified as an inverse problem [14]. Assuming a linear system as described in the previous section, measurements of a wavefront coming from a deformable mirror

can be modelled:

$$\mathbf{s} = \mathbf{\Gamma}\phi \tag{2.2.29}$$

where $\mathbf{s}$ represents a set of spatial derivatives of the incoming wavefront phase, $\phi$ is the corresponding phase (or optical path difference) on the deformable mirror that is being measured, $\mathbf{\Gamma}$ carries the interaction relationship between the mirror signals and the sensor and depends on the sensor geometry.

In practice, for a closed loop system, the interaction matrix $\mathbf{\Gamma}$ expresses the response of the sensor elements to the DM actuator states, and can be produced through a calibration process by applying a unit voltage to each actuator (poking) and recording the corresponding sensor element response. There are a number of wavefront sensor geometries that can be used to derive the phase variations (gradients) on the sensor in wavefront reconstruction [18], but for the purpose of this study we will consider the Fried geometry [23] for a Shack Hartmann WFS, illustrated in figure 2.9. According to Fried's model, the wavefront gradients can be expressed in

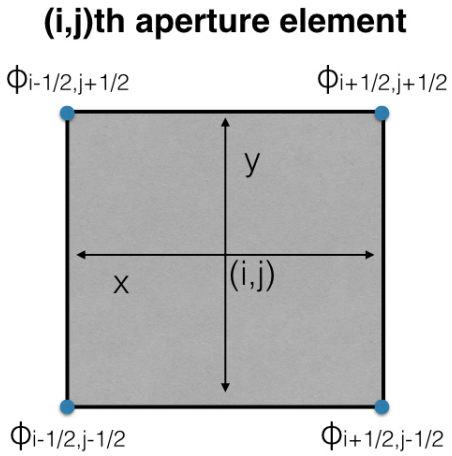



Figure 2.9: Fried geometry

relation to discrete phase values on a grid defined by square telescope subapertures mapped on the sensor area. In this mapping, the phase estimates are evaluated at the four corners of each subaperture, and the gradients are measured for the area of each square. For a subaperture spacing of 1, the resulting relationship can be written:

$$\mathbf{s^x}_{i,j} = \frac{1}{2}\left[\left(\phi_{i+\frac{1}{2},j+\frac{1}{2}} + \phi_{i+\frac{1}{2},j-\frac{1}{2}}\right) - \left(\phi_{i-\frac{1}{2},j+\frac{1}{2}} + \phi_{i-\frac{1}{2},j-\frac{1}{2}}\right)\right] \tag{2.2.30}$$

$$\mathbf{s^y}_{i,j} = \frac{1}{2}\left[\left(\phi_{i+\frac{1}{2},j+\frac{1}{2}} + \phi_{i-\frac{1}{2},j+\frac{1}{2}}\right) - \left(\phi_{i+\frac{1}{2},j-\frac{1}{2}} + \phi_{i-\frac{1}{2},j-\frac{1}{2}}\right)\right] \tag{2.2.31}$$

The problem is defined as estimating the phase values $\phi$ at the corners of subapertures, given the values of the measured gradients $\mathbf{s^x}, \mathbf{s^y}$. Since there are more measurements than unknowns, the least squares method is used to solve the problem by minimising the measurement error $\epsilon_s$. If terms are expressed in matrix form

$$\epsilon_s = \|\mathbf{s} - \mathbf{\Gamma}\phi\|^2 \tag{2.2.32}$$

The solution is found by inverting the interaction matrix. Matrix $\mathbf{\Gamma}$ is not square so its pseudo-inverse is calculated instead. This is equivalent to applying a least square solution for an overdetermined system of equations. In principle, the Moore-Penrose pseudo inverse provides the solution [22] :

$$\left(\mathbf{\Gamma^T}\right)\phi = \mathbf{\Gamma^T s} \tag{2.2.33}$$

$$\mathbf{R} = \left(\mathbf{\Gamma^T \Gamma}\right)^{-1}\mathbf{\Gamma^T} \tag{2.2.34}$$

where $\mathbf{R}$ is the reconstruction matrix.

In practice however, the square matrix $\mathbf{\Gamma^T\Gamma}$ is singular because the overall constant component of the wavefront (piston) cannot be sensed. In addition, there are a number of poorly sensed aberrations in the presence of noise, which induce instability in the system. An example of such aberrations is the so-called "waffle mode" that occurs in the Fried geometry. This mode appears as a checkerboard-like pattern over the aperture, an astigmatic mode that is repeated at the Shack-Hartmann wavefront sensor spatial frequency [24]. Because it produces zero average slope, it yields zero response in the Fried least-squares method and consequently affects the system's imaging performance. Waffle modes are caused commonly by DM - WFS geometrical misregistration and very high sensor noise levels. Under these conditions, a different method is required in order to remove the singularity and filter out unstable components. Singularity removal is typically accomplished in AO using the Singular Value Decomposition (SVD) technique [2]. The SVD is a matrix factorisation that decomposes a matrix to its singular vectors and singular values. A real valued $m \times n$ matrix $\mathbf{M}$ has a decomposition of the form:

$$\mathbf{M} = \mathbf{U\Sigma V^T} \tag{2.2.35}$$

where $\mathbf{U}$ and $\mathbf{V}$ are the $m \times m$ and $n \times n$ orthogonal matrices containing the left and right singular vectors of $\mathbf{M}$ respectively. In our case the columns of $\mathbf{V}$ define the modes of the actuator space, and $\mathbf{U}$ describe the normalised sensor response vectors to each of the modes (to be revised). Matrix $\mathbf{\Sigma}$ is a $m \times n$ diagonal matrix whose main diagonal contains the singular values of $\mathbf{M}$. Using this factorisation we

can then compute the matrix inverse from:

$$\mathbf{M}^{+} = \mathbf{V}\mathbf{\Sigma}^{+}\mathbf{U}^{\mathbf{T}} \tag{2.2.36}$$

Since $\mathbf{\Sigma}$ is diagonal, $\mathbf{\Sigma}^{+}$ is found by replacing every value on the diagonal with its reciprocal. For an inverse problem such as wavefront reconstruction, the singular values of the measurement matrix are a measure of the ill-posedness of the problem. The condition number $k$ of the function that is mapped in the matrix measures the response of the output to small changes in the input variables, in other words, it expresses the stability of the linear system. The condition number is related to the singular values of the linear system matrix, that is to be solved using least-squares with the Euclidean norm, with the relationship:

$$k(\mathbf{M}) = \left(\frac{s_{max}(\mathbf{M})}{s_{min}(\mathbf{M})}\right) \tag{2.2.37}$$

where $s_{max}(\mathbf{M})$ and $s_{min}(\mathbf{M})$ are the maximum and minimum singular values of $\mathbf{M}$. A large condition number indicates an ill-posed - or ill-conditioned - problem that is numerically unstable with limited accuracy. It follows from (2.2.37) that if $s_{min}(\mathbf{M})$ is very small, or close to numerical zero the condition number tends to infinity and the system can't produce a reliable solution. The SVD helps us to treat this problem by providing the singular values matrix which we can then use to filter out the very small values before we calculate the pseudo-inverse. In doing this, we replace the filtered values in $\mathbf{\Sigma}^{+}$ with zeros. This process is also referred to as Truncated Singular Value Decomposition (TSVD).

The TSVD is one form of regularisation for ill-conditioned systems. A variant of this approach exists with the application of the Tikhonov filter [2], that provides suppression of small singular values by asymptotically approaching a lower limit. This type of filtering is equivalent to the TSVD, but does not require the SVD to be performed. Equation (2.2.33) with the application of the Tikhonov filter becomes:

$$(\mathbf{\Gamma^T \Gamma} + \alpha \mathbf{I})\, \phi = \mathbf{\Gamma^T s}, \tag{2.2.38}$$

where $\alpha$ is a regularisation parameter that replaces the singular values matrix $\mathbf{\Sigma}^+$ in equation (2.2.36) with a matrix $\mathbf{T}$ such that

$$\mathbf{T}_{ii} = \frac{\sigma_i}{\sigma_i^2 + \alpha^2} \ . \tag{2.2.39}$$

The above formulations assume a closed loop system model, where the DM and wavefront sensor interact directly, with the DM located before the WFS in the optical path. In the case where the WFS receives the signal directly, independently from the DM, then the system model is configured in open-loop control operation [13]. Open-loop control has the disadvantages that it requires very accurate characterisation of the DM, since it's response is not measured by the WFS, and that the measurements are over the full dynamic range of the uncorrected wavefront. In classical AO systems closed-loop control can be used and is preferred over open-loop, however, in many of the new AO system designs for ELTs, closed-loop is not possible and these have to employ open-loop control.

#### 2.2.2.1 Optimal Solver

Singular value filtering is not capable of producing very high accuracy, and cannot guarantee the removal of all poorly sensed optical effects such as the waffle mode. This is because the actuator mode space, although it is filtered, remains unchanged. To optimise the system, regularisation should be applied in such a way that the unwanted components are suppressed. This can be done using statistical weighting, whereby low weighting is given to such components. Furthermore, higher accuracy is required for atmospheric tomography, as it is an even more ill-conditioned problem, so the least squares fitting method may not be adequate.

The minimum variance solver is an optimal solver for adaptive optics, because the merit function is the residual wavefront variance, rather than the DM command error vector [25–27], and incorporates wavefront phase and measurement noise statistics. As this method estimates the actual wavefront phase values and not directly the DM commands, the WFS sensor measurements model, is in open-loop:

$$\mathbf{s} = \mathbf{\Gamma}\phi + \mathbf{n} \tag{2.2.40}$$

If we assume that both $\phi$ and $\mathbf{n}$ are random vectors with zero mean and covariance $C_\phi$ and $C_n$ respectively, the minimum variance estimator is given by:

$$\hat{\phi} = \hat{\mathbf{R}}\mathbf{s} \tag{2.2.41}$$

where the reconstruction matrix $\hat{\mathbf{R}}$ minimises the variance of the residual wavefront

phase:

$$\hat{\mathbf{R}} = \underset{\mathbf{R} \in \mathbb{R}^{\mathbf{m}\times\mathbf{n}}}{\arg\min}\, E(\|\mathbf{Rs} - \phi\|^2). \tag{2.2.42}$$

The symbol $E$ denotes the expectation value. The minimum variance reconstructor is found in a closed form [14]:

$$\hat{\mathbf{R}} = (\mathbf{\Gamma^T C_n^{-1} \Gamma} + \mathbf{C_\phi^{-1}})^{-1} \mathbf{\Gamma^T C_n^{-1}}. \tag{2.2.43}$$

This solution also treats the problem of instability in high spatial frequencies (such as the waffle mode) of the classical least squares technique by applying smooth regularisation against noise, and weighting the estimate with prior turbulence statistics.

## 2.3 Atmospheric Tomography

The estimation of a three-dimensional phase perturbation profile is referred to as atmospheric tomography [28] and is aiming at correcting the wavefront over a wide field of view for ELTs. A wider AO-corrected field of view is desirable in order to allow high resolution observations of extended objects, but it is also needed to increase sky coverage for reference sources. The main limitation of classical AO that drives the new designs is its inability to deal with the problem of anisoplanatism both for natural and artificial reference sources, and volume reconstruction can be applied to overcome this limitation.

### 2.3.1 Angular Anisoplanatism and sky coverage

Angular anisoplanatism refers to the situation where two sources on the sky that are separated by an angular distance follow different paths through the atmosphere to a telescope aperture plane (figure 2.10). Classical adaptive optics systems correct for the phase distortions that are accumulated at the pupil plane by collecting and analysing light from a distant source in one direction. As a result, there is a phase difference between the corrected wavefront and the actual wavefront that increases with the angular separation between the scientific target object and the reference source. The area within which the mean square wavefront error does not exceed 1 square radian is known as the isoplanatic angle or isoplanatic patch [29].

$$\theta_0 = 0.314 \frac{r_0 \cos \gamma}{\overline{h}} \tag{2.3.44}$$

where $\overline{h}$ is the average turbulence height and $\gamma$ is the zenith angle.

This angle ranges from about 2 arcseconds in the visible wavelengths up to about 10 arcseconds in the near-infrared [13], and the observed astronomical object must be within this distance from the reference source in order to get adequate AO correction. As a consequence, the part of the sky that can be seen with adaptive optics, is limited by the proximity of the objects to a bright enough guide star.

### 2.3.2 Focal Anisoplanatism

To overcome the limitations of sky coverage imposed by the availability of natural guide stars and angular anisoplanatism, laser beacons can be used to project a bright

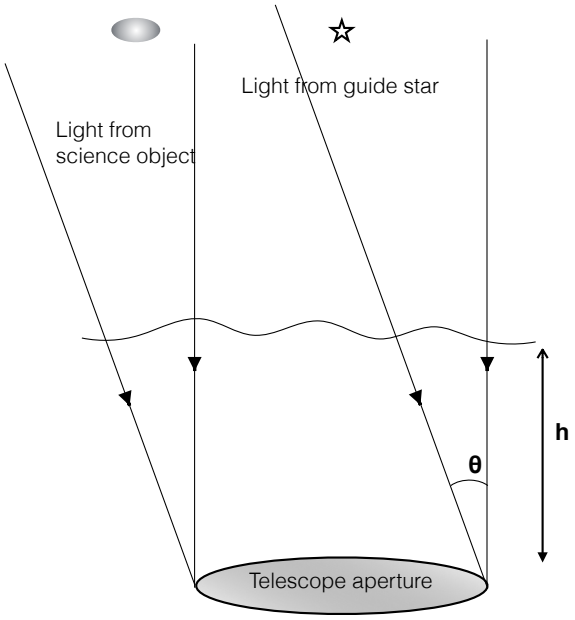



Figure 2.10: Angular Anisoplanatism

point source on the sky near the object of interest. These are most commonly known as laser guide stars and can be produced by Rayleigh scattering at about 16 - 20 km altitude in the atmosphere, or by exciting atoms at the mesospheric sodium layer at 80 - 90 km [18]. Due to the finite distance of the artificial stars from the telescope aperture, only the conical section defined by the source and the aperture can be sensed to measure the turbulence, while the light coming from the observed object that can be thought of to be at an infinite distance traverses through a cylindrical area the size of the telescope diameter (figure 2.11), therefore, some of the turbulent phase information is lost. This is called focal anisoplanatism or cone effect, and it induces a further residual wavefront error to an adaptive optics system [30]:

$$\sigma_{fa}^2 = \left(\frac{D}{d_0}\right)^{5/3} \tag{2.3.45}$$

where $D$ is the telescope diameter and $d_0$ is called the characteristic distance. The term $d_0$ varies with the observing wavelength $\lambda$, the LGS altitude $z_{LGS}$, and the turbulence $C_n^2(z)$ profile over atmospheric altitude $z$, at zenith angle $\psi$. It can be defined as the diameter over which $\sigma_{fa}^2$ is 1 rad$^2$. An expression for $d_0$ is given by Tyler [31]:

$$d_0 = \lambda^{6/5} \cos^{3/5} \psi \left[ 19.77 \int \left( \frac{z}{z_{LGS}} \right)^{5/3} C_n^2(z)\, \mathrm{d}z \right]^{-3/5} \tag{2.3.46}$$

Given the fixed altitudes where artificial guide stars are placed, for a given atmospheric profile, the cone effect becomes worse as the telescope diameter increases.

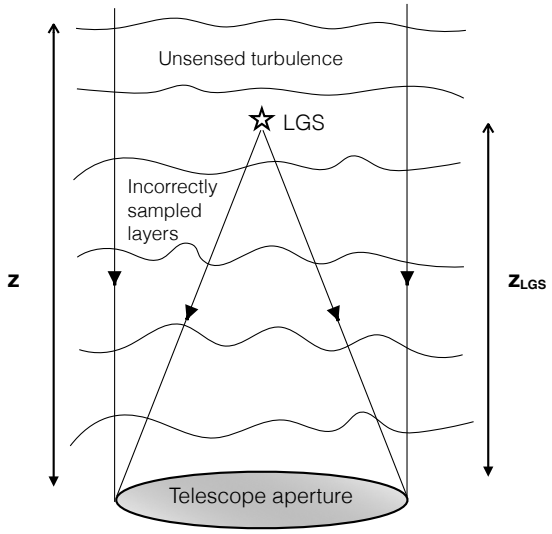



Figure 2.11: Focal Anisoplanatism

### 2.3.3 Volume Reconstruction

By using information from more than one laser guide stars pointing at different directions, we can apply AO correction to the volume of atmospheric turbulence, which partially compensates for angular and focal anisoplanatism [32–34].

Assuming a geometrical beam propagation model and a layered atmosphere with L layers of turbulence as in figure 2.12, when light from a guide star at angle $\theta$ travels through layers of turbulence, the wavefront phase $\phi(x, y, \theta)$ at the pupil plane is found by accumulating the phases $\psi(x, y, \theta, z_l)$ from the corresponding layers along the propagation path.

$$\phi(x, y, \theta) = \sum_{l=1}^{L} \psi((x, y) + z_l\theta, z_l) \tag{2.3.47}$$

Measurements from M guide stars using a wavefront sensor for each guide star, can be modelled:

$$\mathbf{s} = \mathbf{G}\psi + \mathbf{n} \tag{2.3.48}$$

Here, $\mathbf{s}$ and $\psi$ are block vectors that represent the total of measurements for all guide stars and the phase values at all layers respectively, with each block corresponding to each guide star and atmospheric layer. Matrix $\mathbf{G}$ also has the form of a block matrix that maps linearly the contribution of atmospheric phase values on all layers to wavefront sensor measurements in all directions, and $\mathbf{n}$ is the block vector of measurement noise for all wavefront sensors.

The system of equations formed, following the classical AO reconstruction case, has $2 \times N_{sub} \times N_{star}$ equations with $M \times N_{phase}$ unknowns. Solving with the classical

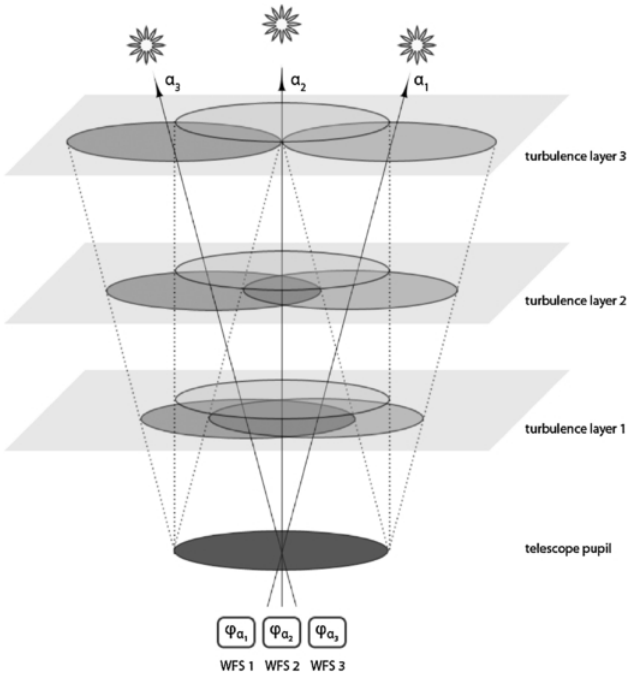



Figure 2.12: Atmospheric tomography model (credit: Ramlau and Rosensteiner [35])

least squares and matrix inversion approach as in (2.2.34) requires the inversion of a matrix of size $N_{equations} \times N_{unknowns}$ at initialisation time, and a large matrix-vector multiplication in real-time. The global solution of the tomographic reconstruction problem for ELTs will require solving a very high order system of equations, and computationally efficient methods suitable to the problem size and nature are needed.

## 2.4 Extremely Large Telescopes

Adaptive Optics for Extremely Large Telescopes will employ new configurations that will address the problem of sky coverage for reference sources of appropriate brightness, with techniques for widening the field of view for the corrected wavefront, selecting multiple targets with small fields of view, or correcting focal anisoplanatinsm for laser guide stars. There are several configurations currently planned

[36], and in the following sections I describe the two principle designs that are of interest in this study.

### 2.4.1 Multi-Conjugate Adaptive Optics

MCAO [28, 37] is a technique aimed primarily at increasing the corrected field of view of adaptive optics. MCAO will address the problem of angular anisoplanatism in the corrected field of view of adaptive optical systems by optically conjugating several deformable mirrors to different heights that correspond to turbulent layers in the atmosphere. Multiple laser guide stars can be used for wavefront sensing at different directions, and this can also compensate for the cone effect. The concept is illustrated in figure 2.13.

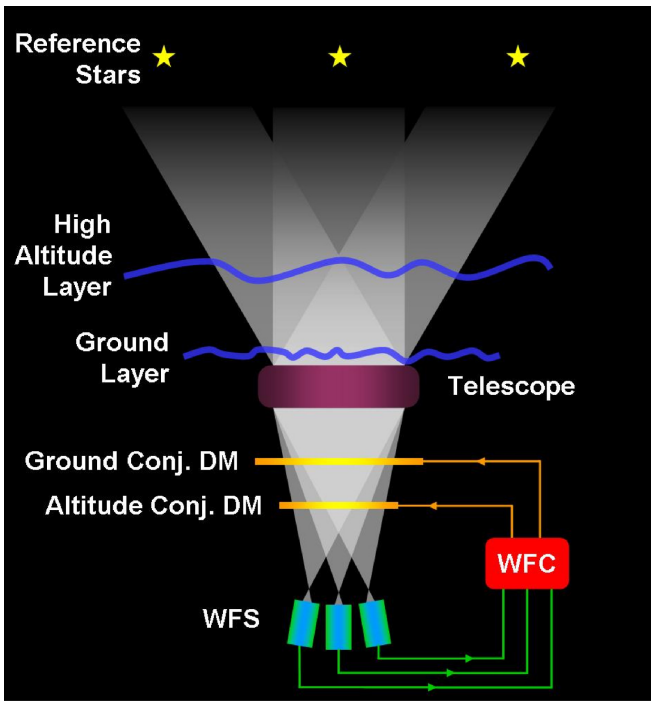



Figure 2.13: MCAO concept (Image ©ESO).

In order to probe the atmospheric layers, MCAO is used with atmospheric to-

mography to provide enough information over the desired field of view and also in order to accurately solve for many turbulence layers corrected by their conjugated DMs. The tomography solver of equation (2.3.48) will then need to be fitted to the number of DMs according to their conjugation height.

### 2.4.2 Multi-Object Adaptive Optics

The objective in Multi-Object Adaptive Optics (MOAO) design is to observe multiple small size targets within a large field. This is beneficial for surveys of distant galaxies with multi-object spectroscopy, where a large patrol field is required [38]. MOAO was first introduced with the FALCON concept [39] in 2001, and since then it has evolved to a technique that is central to the design of the Multi-Object Spectrograph for Astrophysics, Intergalactic-medium studies and Cosmology (MOSAIC) instrument [40], which is to be employed at the E-ELT (formerly known as ELT Adaptive Optics for GaLaxy Evolution (EAGLE) [41]).

MOAO, like MCAO, uses tomographic reconstruction to determine the DM correction that is required over a wide field of view, but instead of applying this correction to the entire field, it corrects only along multiple individual lines of sight using a single DM for each, in a limited field of view that contains the object of interest. The concept is illustrated in figure 2.14.

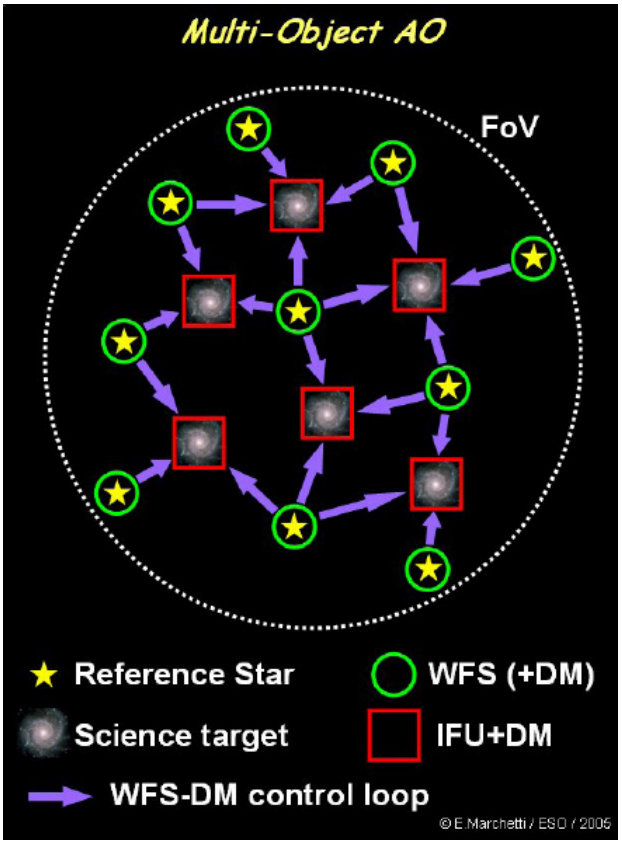



Figure 2.14: MOAO concept (Image ©ESO).

Because the DMs are pointing to different directions than the WFSs, the system must operate in open-loop, with the associated challenges as mentioned previously.

# Chapter 3

# Advanced Algorithms for Real-time Control Solvers

In this chapter I review novel methods that have been proposed for the solution of the Adaptive Optics inverse problem on Extremely Large Telescopes from the perspective of computational performance. This review expands in more detail on those algorithmic methods that were of interest to this study. I discuss the advantages and disadvantages of those methods and their applicability to massively parallel architectures.

## 3.1 Tomographic Reconstruction on ELTs

The problem of the practical implementation of tomographic reconstruction on ELTs is three-fold.

First, due to the increased telescope diameter, the order of wavefront measurements and DM actuators is very high, such that if the system is solved with the

classical least squares approach and the application of noise filtering with the TSVD, it will be unstable because of the increased number of poorly seen or unseen aberrations [24, 42, 43]. To regulate the system against noise, there is a need to use prior statistical information for turbulence and noise, hence there is a need to use an optimal solver.

Second, the application of a direct inversion of a matrix with up to $10^5 \times 10^4$ elements that would be needed in order to use the standard matrix-vector multiplication technique requires a very large amount of computational power and memory storage, and the execution times for matrix-vector multiplications of this scale exceed by far the typical requirements of AO systems. It is desirable to employ new computational algorithms that reduce the amount of memory and computations required to solve the system.

The third point adds to the second one the application of parallelisation, since the trends in computing power increase are in parallel processing. Since the last decade, there has been rapid advancement in the development of parallel processing hardware, from multi-core CPUs to dedicated and special-purpose hardware accelerators, and it is essential to adapt software algorithms to exploit parallelism in computing devices. To accomplish that, there is a need for both, algorithms that are suited as much as possible to parallel processing, and methods to map these to the specific hardware.

### 3.1.1 Advanced Computational Algorithms

Several methods have been proposed to reduce the computational demands that standard matrix inversion algorithms have, by producing a reconstruction matrix with sparse techniques, by using iterative solvers that also make use of sparse approximations of the operators involved in the reconstructor, and by a state-space model predictive controller approach.

A minimum variance reconstructor with Cholesky factorisation techniques, using a sparse approximation to the inverse Kolmogorov phase covariance matrix for MCAO has been proposed by Ellerbroek [26]. However, its computing performance potential is limited due to the influence that multiple guide star and turbulent layers have on the sparsity of their phase-to-WFS interaction matrix operator [26, 44].

Iterative methods based on the Preconditioned Conjugate Gradients (PCG) have also been studied: a Multi-Grid Preconditioned Conjugate Gradients (MGPCG) [45, 46], a Fourier Domain Preconditioned Conjugate Gradients (FDPCG) [47, 48], and a Fractal Iterative Method (FrIM) [49]. The first two have been developed in the context of the TMT [50], and the third for the E-ELT case, and all have demonstrated results in simulations.

The FDPCG solves the problem in two steps, a tomography estimation step which is the most computationally demanding, and a DM fitting step. The estimation step treats the wavefront measurement process as a spatial filtering operation that can take advantage of Fourier domain techniques. The wavefront sensing operator is the product of three terms: geometric propagation of the guide stars through the atmosphere, wavefront gradient sensing of the pupil plane, and aperture mask-

ing. In the Fourier domain, the first two are shift operators that are transformed to diagonal matrices using the Fourier shift theorem, and the third corresponds to a convolution that becomes point-wise multiplicative using the convolution theorem. The incorporation of priors involves an approximation of the phase covariance matrix that results in a structurally convenient matrix form. These operators are then combined to assemble an efficient Fourier domain preconditioner that allows the conjugate gradients algorithm to converge in very few iterations.

The FrIM creates a preconditioner based on a decomposition of the phase covariance matrix with an operator that is built by exploiting the fractal structure of turbulence using a fast random mid-point displacement algorithm [51] for wavefront generation that results in approximating the phase covariance decomposition. This preconditioner can be applied implicitly using simple iterations on an equation with changed variables.

There is a different category of algorithms that are based on a state-space approach from the AO control model perspective, which take into account the temporal correlation of turbulence between control states, to optimise the result. A Linear Quadratic Gaussian control regulator that is studied has gained increasing popularity [52, 53] and has demonstrated results in simulations [54, 55] and was also recently validated on-sky [56]. Here, the computation consists of two steps, an estimation that fulfils the minimum variance criterion using a Kalman filter [57] and projection to the DM space. The Kalman filtering operation is usually done in a non-iterative fashion by pre-computing a gain matrix, but iterative methods have been considered as well [58].

More recently, a solution using Kaczmarz iteration [35] and a Finite element-wavelet hybrid algorithm [59, 60] have been presented. A hierarchical wavefront reconstructor [61] and a Cumulative Reconstructor with domain decomposition [62] have recently been demonstrated on-sky [63].

## 3.2 Iterative Methods for Minimum Variance Solvers

### 3.2.1 Conjugate Gradients

The conjugate gradient method solves a linear system of equations $Ax = b$ by finding the minimum of the quadratic form taking its gradient to zero. This can be done when the coefficient matrix $A$ is symmetric and positive definite. The zero gradient is approached step by step starting from an initial assumption for the unknown vector values $x$. At every step, the result from a vector that expresses an orthogonal direction $p$ at the matrix space, is added to $x$. This is the conjugate direction, and it is found by conjugating the residual vectors $r$. Vector conjugation is expressed in matrix algebra with the inner dot product.

This solution is efficient when the matrix is sparse and well-conditioned. Because this is not always the case, a preconditioner matrix $C$ can be applied to the original system in a way that the condition number of matrix $A$ can be improved so as to increase the rate of convergence to the solution. The system is then transformed to:

$$C^{-1}Ax = C^{-1}b. \tag{3.2.1}$$

The steps of the Preconditioned Conjugate Gradients [64] are listed in Algorithm 3.1.

**Algorithm 3.1** Preconditioned Conjugate Gradient Method.

$r_0 = b - Ax_0$

$z_0 = C^{-1}r_0$

$p_0 = z_0$

**for** $k = 0$ until convergence **do**

$q_k = Ap_k$

$a_k = \dfrac{r_k z_k}{q_k p_k}$

$x_{k+1} = x_k + a_k p_k$

$r_{k+1} = r_k - a_k q_k$

$z_{k+1} = C^{-1} r_{k+1}$

$\beta_k = \dfrac{r_{k+1} z_{k+1}}{r_k z_k}$

$p_{k+1} = z_{k+1} + \beta_k p_k$

**end for**=0

$C$ has to approximate $A$ in order to improve convergence and to reach an acceptable solution in a small number of iterations.

### 3.2.2 Fourier Domain Preconditioned Conjugate Gradients Method

The Fourier Domain Preconditioned Conjugate Gradients [47, 48] is an iterative algorithm for the computation of the atmospheric tomography solution in MCAO. Based on the modelling of wavefront measurements on the periodic, spatial frequency domain, it attempts to create a very sparse approximation to the reconstruction problem, that can then be solved efficiently using the PCG.

FDPCG implementation assumes an open loop, linear system model, and an op-

timal minimum variance controller. The atmosphere is modelled as a set of discrete layers with a Kolmogorov refractive index distribution, and the phase perturbations are considered as the result of geometric optics propagations from point sources at various angles to the pupil-plane. The wavefront measurements are modelled according to Fried as discrete approximations to the phase gradient. The minimum variance reconstruction matrix is derived from previous works [26] as follows:

$$s = G\psi + n \implies \psi = E_{MV}s \tag{3.2.2}$$

$$E_{MV} = (G^T C_n^{-1} G + C_\psi^{-1})^{-1} G^T C_n^{-1}, \text{ where } G = M\Gamma P. \tag{3.2.3}$$

In the above notation, $\psi$ is the vector of discrete phase values at the conjugate turbulent layers, $s$ is the vector of pupil-plane measurements. The operators $G, C_n, C_\psi$ represent influence of atmospheric volume phase $\psi$ to WFS measurements, noise covariance and phase covariance respectively corresponding to the discrete layers. $G$ is the product of a circular aperture masking operator $M$, of the mapping operator of pupil plane phase to WFS measurements $\Gamma$, and of the operator $P$, that represents geometrical propagation of a beam from a guide star to the telescope aperture.

If we set $A = G^T C_n^{-1} G + C_\psi^{-1}$ and $b = G^T C_n^{-1} s$ the problem is formed as a Symmetric Positive Definite (SPD) linear system of equations that is suitable for solving with the PCG method.

The essence of the FDPCG algorithm for use in the minimum variance tomographic solver is finding a preconditioner for $A$ in the Fourier domain. This is motivated by previous work that shows the wavefront sensing operation can be

modelled as a spatial filter [65] in the discrete frequency domain, and that the numerical operators involved result in very sparse representations, while retaining an acceptable accuracy for the mathematical turbulence model. In addition, the use of Fast Fourier Transforms (FFTs) allows for a fast calculation of these terms.

#### 3.2.2.1 Preconditioning in the Fourier Domain

The resultant Fourier domain preconditioner is a sparse $L \times L$ block matrix where $L$ is the number of atmospheric layers, each consisting of an $N \times N$ grid of phase points. This matrix is built by the components of the reconstruction matrix of equation (3.2.3).

1. A propagator $P$ representing guide star propagation from the atmosphere to the pupil plane, depending on view angle , and lateral position on the layer grid $x$, and modelled as in [48]:

$$[P\psi](x,\theta) = \sum_{l=1}^{L} \psi(x + z_l\theta, z_l) \tag{3.2.4}$$

$P$ has the form of a block matrix $P(k,l)$ where block indices $k$ and $l$ denote the $k$th guidestar and the $l$th layer. From the relationship above, considering a given guide star $k$ and layer $l$ and $x$ in one lateral direction we can see that each block $P_{kl}$ of the operator $P$ represents a shift in position in the spatial domain, therefore a shift in phase in the spatial frequency domain. In the discrete Fourier space, according to the shift theorem [66] this corresponds to an element-wise multiplication of vector $\psi$ with $e^{-j\omega\phi}$, $\omega = \frac{2\pi i}{N}$ being the angular spatial frequency for position index $i$ in our grid of $N$ points in one direction

and $\phi = -z_l\theta$. In two dimensions we have a 2-dimensional spatial phase shift in the Fourier domain, adding a second index $j$ to the above and a shift to that direction transforming the multiplicative term to $e^{-j(\omega_i\phi_i+\omega_j\phi_j)}$. This term is calculated by replacing the indices $i, j$ with those of the 2-dimensional vector $\psi$ and the current block indices $k, l$ corresponding to the known $\theta, z_l$. As a result, $P_{kl}$ is transformed to a diagonal matrix and $P$ to a block matrix with diagonal blocks in the Fourier domain. This process can be referred to as a filtering operation [48]. $P$ and its Fourier representer $\hat{P}$ could take the form:

$$P = \begin{bmatrix} P_{00} & \cdots & P_{0l} \\ \vdots & \ddots & \vdots \\ P_{k0} & \cdots & P_{kl} \end{bmatrix}, \; P_{kl} = \begin{bmatrix} 0_{0,0} & \cdots & \cdots & 1_{0,s} & \cdots & 0_{0,n-1} \\ \vdots & 0_{1,1} & \cdots & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & 0_{i,j} & \cdots 1_{i,j+s} & \\ 1_{i+1,0} & & & & & 0_{i+1,n-1} \\ \vdots & \ddots & \vdots & & \ddots & \vdots \\ 0_{n-1,0} & \cdots & 1_{n-1,n+s-1} & \cdots & \cdots & 0_{n-1,n-1} \end{bmatrix}$$

$$\hat{P} = \begin{bmatrix} \hat{P}_{00} & \cdots & \hat{P}_{0l} \\ \vdots & \ddots & \vdots \\ \hat{P}_{k0} & \cdots & \hat{P}_{kl} \end{bmatrix}, \hat{P}_{kl} = \begin{bmatrix} f_0 & 0 & \cdots & \cdots & 0 \\ 0 & f_1 & 0 & \cdots & \vdots \\ \vdots & 0 & \ddots & \ddots & \vdots \\ 0 & & \cdots & f_n & \end{bmatrix} \tag{3.2.5}$$

Where $i, j$ are the random indices corresponding to a position on the grid of $n = N \times N$ phase points at layer $l$, $s$ is the position displacement on the pupil plane due to guide star direction and layer height, the $\hat{}$ accent shows Fourier representation, and $f_n$ is the multiplicative factor that applies the filtering in the Fourier domain.

2. The pupil mask $M$, representing the masking of the rectangular computational grids by a circular aperture. In a $N \times N$ rectangular grid of discrete points this operator has the value of 1 if the points lie within the aperture and zero if they are outside. In relation to the wavefront sensing operation this can be expressed as in [47]:

$$[Ms](x) = \begin{cases} s(x), & \text{if } x \in \Omega \\ 0, & \text{otherwise} \end{cases} \tag{3.2.6}$$

where $\Omega$ is the circular aperture grid area and $s(x)$ the output of the wavefront sensor corresponding to point $x$ on the grid.

In the spatial domain, this masking operation is a point-wise multiplication of the grid elements with the equipositioned mask elements. In the Fourier space, using the Convolution Theorem [66], the operation transforms to a convolution. Assuming the grid is periodic with period $N$ in each direction, a two-dimensional circular convolution is appropriate to account for edge points. For two functions $f$ and $g$ this relationship is described as [66]:

$$f(m,n) = (h \circledast \circledast g)(m,n) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} h((m-k, n-l) \mod N) g(k,l), \tag{3.2.7}$$

where $\circledast$ denotes circular convolution.

Being a linear operation, the above can be expressed in matrix-vector multiplication form. The 2-D signal $g(m,n)$ is written as a $N \times N$ row ordered vector $\mathbf{g}$, $h$ as a column block vector $\mathbf{h}$ with blocks corresponding to the rows

of $h(m,n)$ and the convolution is applied by the $NN \times NN$ matrix $\mathbf{H}$, as in the representations of (3.2.8), (3.2.9), and (3.2.10) :

$$\mathbf{f} = \mathbf{Hg} \rightarrow \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N^2-1} \end{bmatrix} = \begin{bmatrix} H_0 & H_{N-1} & \cdots & H_1 \\ H_1 & H_0 & \cdots & H_2 \\ \vdots & H_1 & \ddots & \vdots \\ H_{N-1} & \cdots & H_1 & H_0 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{N^2-1} \end{bmatrix} \tag{3.2.8}$$

$$H_j = \begin{bmatrix} h_{j,0} & h_{j,N-1} & \cdots & h_{j,1} \\ h_{j,1} & h_{j,0} & \cdots & h_{j,2} \\ \vdots & h_{j,1} & \ddots & \vdots \\ h_{j,N-1} & \cdots & h_{j,1} & h_{j,0} \end{bmatrix}, \tag{3.2.9}$$

$$j = 0, 1, \cdots, N-1, \quad h = \begin{bmatrix} [ & h_{0,0} & \cdots & h_{0,N-1} & ] \\ [ & h_{1,0} & \cdots & h_{1,N-1} & ] \\ [ & \vdots & \cdots & \vdots & ] \\ [ & h_{N-1,0} & \cdots & h_{N-1,N-1} & ] \end{bmatrix} \tag{3.2.10}$$

Matrix $H_j$ is arranged as a circulant matrix, where row elements are successively rotated to the right, and represents the 1-D convolution operation of each row vector of $\mathbf{h}$. The structure occurs due to the nature of the convolution operation. Matrix $\mathbf{H}$ has this property in a block form (due to the second dimension in the convolution) with its rows of blocks rotated to the right, and because $\mathbf{H}$ consists of blocks $H_j$, it is called a Block-Circulant with Circulant Blocks (BCCB) [2] matrix. If we replace $\mathbf{h}$ with the Fourier transformed mask signal $M$, then the Fourier representer for the masking operation is a matrix of the form of $\mathbf{H}$, which will be noted by $\hat{M}$. The Fourier transform of the

pupil mask is an Airy pattern [47] and is a dense matrix. To obtain a sparse approximation to the convolution operator $\hat{M}$, the transformed mask is truncated to a few points around the central core of the Airy pattern. In matrix form, the maximum of the Airy function occurs at the upper left most index of the array (for example $h_{00}$ in the above illustration), so the approximation to $\hat{M}$ results in a matrix with diagonal bands. If truncation is done to the maximum point, then $\hat{M}$ is diagonal.

3. Operator $\Gamma$, separated in $x$ and $y$ direction, producing $\Gamma_x$ and $\Gamma_y$, maps pupil plane phase to WFS measurements. The mapping follows the Fried geometry of discrete phase gradients. In the same way as in 1, with the use of the Fourier shift theorem, $\Gamma_x$ and $\Gamma_y$ are transformed to diagonal matrices in the Fourier domain.

4. Inverse phase and noise covariance. The phase covariance can be represented as a block diagonal matrix with each block corresponding to one atmospheric layer. For a Kolmogorov power spectrum the inverse phase covariance for each layer forms a discrete function of the magnitude of the wavenumber $|k|$ in the Fourier domain [67]. Expanding the 2-dimensional wavenumber vector in matrix form, the resultant inverse phase covariance Fourier representer is a block diagonal matrix with diagonal blocks $[\hat{C}_{\psi}^{-1}]_l$, where $l$ is the layer index. The noise covariance matrix can be represented by the scalar variance of the sensor noise, $\sigma^2$ [14].

Gathering the above operators together to form an approximation to matrix $C$ in the Fourier domain, we have for every layer-indexed block $l, m$ :

$$[\hat{C}]_{lm} = \sigma^{-2} \sum_{j=1}^{N} \hat{P}_{jl} * (\hat{\Gamma}_x * \hat{M} * \hat{M}\hat{\Gamma}_x + \hat{\Gamma}_y * \hat{M} * \hat{M}\hat{\Gamma}_y)\hat{P}_{jm} + \delta_{lm}[\hat{C}_{\psi}^{-1}]_l \quad (3.2.11)$$

where $*$ denotes conjugate transpose, [ ] denotes block, and $\hat{}$ denotes the Fourier representation. $\delta_{lm}$ is the Kronecker delta for $l = m$.

Keeping in mind all the above results of the Fourier transforms, and using the diagonal approximation to $\hat{M}$, the resulting matrix is an $L \times L$ block with $N \times N$ blocks sparse band matrix, where $L$ is the number of layers and $N$ is the number of phase points in each layer. If reordering is applied, we can have all bands around the main diagonal. $\hat{C}$ will have the form:

$$\hat{C} = \begin{bmatrix} \hat{C}_{0,0} & \cdots & \hat{C}_{0,L-1} \\ \vdots & \ddots & \vdots \\ \hat{C}_{L-1,0} & \cdots & \hat{C}_{L-1,L-1} \end{bmatrix}, \quad \hat{C}_{lm} = Diag\left\{c_{lm_0}, c_{lm_1}, \cdots, c_{lm_{N-1}}\right\}, \quad (3.2.12)$$

and, after reordering:

$$\hat{C} = BlockDiag\left\{\hat{C}_{ll}\right\} \quad (3.2.13)$$

The size of $\hat{C}_{ll}$ is small, equal to the number of blocks when the computational grids are equally spaced [48]. If we have an atmospheric model with $L$ layers, then $\hat{C}_{ll}$ is an $L \times L$ array. Using two-dimensional Fourier transforms and one sparse matrix-vector multiplication in the preconditioning step, the algorithm converges in as little

as 3 iterations, with a cost per iteration of $\mathcal{O}(n \log n)$, where $n$ is the order of the system.

### 3.2.3 Fractal Iterative Method

First proposed in 2007 [68], the FrIM also uses PCG iterations to find the minimum variance solution for atmospheric tomography. As mentioned in 2.2.2.1 the preconditioner is based on a decomposition of the phase covariance matrix with an operator that is built by exploiting the fractal structure of turbulence using a fast random mid-point displacement algorithm [51] for wavefront generation that results in approximating the phase covariance decomposition.

Recalling the minimum variance solution:

$$\phi = (G^T C_n^{-1} G + C_\phi^{-1})^{-1} G^T C_n^{-1} s \tag{3.2.14}$$

and adapting to the conjugate gradient form of equations $A\phi = b$, we have:

$$A = G^T C_n^{-1} G + C_\phi^{-1} \text{ and } b = G^T C_n^{-1} s \tag{3.2.15}$$

As seen in the previous section, operators $G$ and $C_n$ are sparse and can be inexpensive to apply. Operator $C_\phi$ however, that is the covariance matrix of the wavefront for Kolmogorov turbulence, is not sparse and both the inversion and the application of this matrix to the conjugate gradients iterations would be costly hence the need for a sparse approximation method for this operator. FrIM looks at

decompositions of the form:

$$C_\phi = KK^T \tag{3.2.16}$$

where $K$ is a square invertible matrix which is designed to operate as a wavefront generator for the turbulence phase structure function, from a set of independent random variables $u$, such that $\phi = Ku$. Following the rules of the mid-point algorithm, with some modifications, an approximation of K can be constructed in $\mathcal{O}(N)$ operations, where $N$ is the number of random variables used to produce the wavefronts.

#### 3.2.3.1 Fractal Wavefront Generation Model

A Kolmogorov phase screen has the property of self-similarity, that is, it appears similar in every scale it is viewed, and it is also a fundamental principle in fractal analysis. This property is a consequence of the power law in the turbulence phase structure function [51].

Lane et al. [51] have developed a fast method for generating a Kolmogorov phase screen, making use of an algorithm that originates from computer graphics applications for terrain generation, first introduced in [69].

The random mid-point displacement algorithm follows a refinement process that generates height values on a 2-dimensional grid, starting from 4 seeds at the corners of an initial square, and successively sub-dividing each square into 4 smaller squares by producing the additional values from the linear interpolation of their neighbouring points. It then adds a random value to the point at the centre of the initial square (the mid-point), which is generated so that the variance follows an appropriate law for the simulated surface. Lane et al. adjust the process so that the starting

points are Kolmogorov turbulence samples instead of Gaussian random variables, enabling it to conform to the Kolmogorov structure function that otherwise proved problematic [51, 70]. Figure 3.1 illustrates the process.

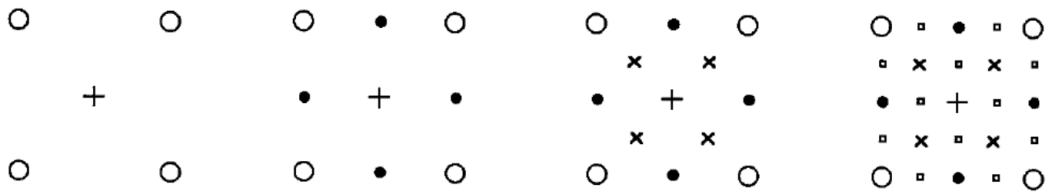


Figure 3.1: 4 steps of the mid-point displacement algorithm (Lane et al. 1992)

Tallon et al. [68] modified this algorithm further in order to construct a sparse fractal operator that produces the regularisation term in 3.2.14, which can also be used to build a preconditioner for the conjugate gradients iterations, with respect to the variable $u$ [49]. The steps of the mid-point algorithm are modified as shown in figure 3.2. First the initial wavefront values are produced from 4 random variables instead of 6 in order to maintain an equal size for the random and wavefront values in the relationship $\phi = Ku$, that will ensure $K$ is invertible. Values on the edges are generated from 3 neighbouring points on the corners of the triangle formed by the edge corners and the mid-point, while in [51] they are produced by only the 2 edge corners. The interpolation weights are calculated such as to produce wavefronts with the properties of the Kolmogorov turbulence phase structure function (shift invariant and isotropic) [49]. $K$ is constructed based on the covariances of the generated values at each step. It should be noted that the step for producing new edge points shown in [49] is already a feature of the original mid-point algorithm as described in [69, 71].

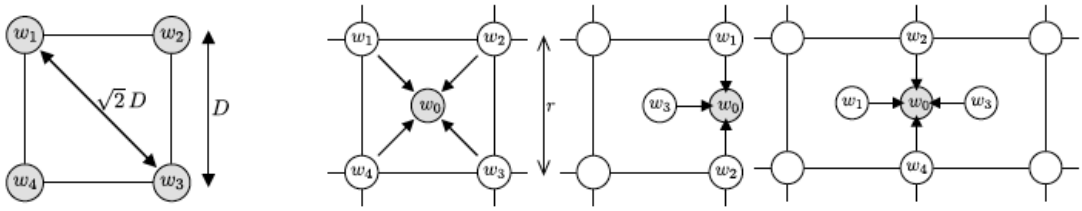



Figure 3.2: 4 steps of the FrIM wavefront generation algorithm (Tallon et al. 2006)

#### 3.2.3.2 Preconditioning with the Fractal Operator

The fractal operator $K$ is related to the wavefront reconstruction with Conjugate Gradients (CG) in equation (3.2.15) through the random variables $u$ that are used to generate the wavefront with a change of variables $\phi = Ku$. Replacing this in 3.2.15 we have

$$(K^T G^T C_n^{-1} GK + I)u = (K^T G^T C_n^{-1})s \tag{3.2.17}$$

Solving the above equation with preconditioner $M = K^T K^{-1}$, we get the solution for $u$ and the wavefront values are obtain by $\phi = Ku$. Convergence is faster when the change of variables is used because $K$ is sparse, and the covariance matrix is replaced by the identity matrix, and the substitution of variables occurs only once in the end.

FrIM has shown results on simulations with MCAO for the E-ELT [72, 73], and more recently, an improved version for atmospheric tomography has been proposed, by the name FrIM-3D [74].

## 3.3 Control-based Methods

These reconstruction methods view the problem from a control theory perspective, and try to derive an optimal control law that predicts the system state that contains the residual wavefront error. Studies of this approach have suggested Kalman filtering techniques and Linear Quadratic Gaussian Control and can be found in [3, 52, 53, 75–79]. It should be noted that currently the studies of those methods focus on optimal quality of the result and are less concerned with computational performance, although there have been efforts in reducing the computational cost, for example in [53, 58].

### 3.3.1 Linear Quadratic Gaussian Controller

A Linear Quadratic Gaussian Controller for Adaptive optics has the benefits of increased accuracy that provides optimal image performance and better noise suppression. It can also have a specific advantage of the ability to compensate for the effects of vibrations and wind disturbances to telescope structures, as recently demonstrated on-sky [56]. It is based on a state-space model of the system (see section 3.3.1.2) and combines a Linear Quadratic Estimator (LQE), which is applied as a Kalman filter, with a Linear Quadratic Regulator (LQR), where the term Gaussian implies the Gaussian noise statistics. This combination is separable, which means that the LQE and LQR problems can be solved independently.

#### 3.3.1.1 Problem Formulations

As mentioned in chapter 2, the adaptive optics system is modelled as a discrete-time closed-loop process evolving in successive time intervals of the WFS integration time $T$. The following revisits the adaptive optics control system equations under the formalisms of control theory, in order to introduce the optimal control law formulation. It is a general description that can be adapted to fit a system with multiple atmospheric layers, guide stars and wavefront sensors, and deformable mirrors.

In discrete time, the wavefront sensor measurement equation takes the form

$$\mathbf{y_k} = \mathbf{D}\phi_{k-1}^{res} + \mathbf{w_{k-1}}, \qquad \phi_k^{res} = \phi_k^{tur} - \phi_k^{corr} \tag{3.3.18}$$

where $\mathbf{y_k}$ is the measurement vector at time $kT$, $\mathbf{D}$ is the linear response operator of the WFS, $\phi_k^{tur}, \phi_k^{corr}, \phi_k^{res}$ are the turbulent, corrected, and residual phases respectively and $\mathbf{w_k}$ the measurement noise. The DM correction is applied during $[kT, (k+1)T]$ through an influence matrix $\mathbf{N}$ that acts on control voltages $\mathbf{u_k}$ available at time $kT$.

$$\phi_{k+1}^{corr} = \mathbf{N}\mathbf{u_k} \tag{3.3.19}$$

Therefore it follows:

$$\phi_{k+1}^{res} = \phi_{k+1}^{tur} - \mathbf{N}\mathbf{u_k} \tag{3.3.20}$$

The optimal criterion for AO control is the minimisation of the residual phase variance:

$$\mathbf{J}(\mathbf{u}) = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \|\phi_k^{res}\|^2 \tag{3.3.21}$$

From (3.3.20), $\mathbf{u_k}$ can be found by a least-squares minimization

$$\mathbf{u_k} = (\mathbf{N^T N})^{-1}\mathbf{N^T}\phi_{k+1}^{tur} \quad , \tag{3.3.22}$$

which is an orthogonal projection of $\phi_{k+1}^{tur}$ to the DM mode space. The projector operator is defined as:

$$\mathbf{P} = (\mathbf{N^T N})^{-1}\mathbf{N^T} \tag{3.3.23}$$

In practice, $\phi_{k+1}^{tur}$ is not known in complete accuracy, but its estimate $\hat{\phi}_{k+1|k}^{tur}$ is used instead, which is the conditional expectation of $\phi_{k+1}^{tur}$ given a set $\mathscr{I}_k$ of prior information and measurements until time $kT$, that can be computed with a minimum-variance estimator.

$$\hat{\phi}_{k+1|k}^{tur} = \mathrm{E}[\phi_{k+1}^{tur}|\mathscr{I}_k] \tag{3.3.24}$$

Equation (3.3.22) then becomes:

$$\mathbf{u_k} = (\mathbf{N^T N})^{-1}\mathbf{N^T}\hat{\phi}_{k+1|k}^{tur} \tag{3.3.25}$$

The above shows that the optimal control can be computed separately by solving the stochastic minimum variance estimation problem in (3.3.24), and using the estimate to solve the deterministic control problem of (3.3.22).

The priors $\mathscr{I}_k$ are deduced from the turbulent phase model. Spatial and temporal correlations are considered in building the priors. The spatial correlation matrix is

defined as:

$$\Sigma_\phi = \mathrm{E}[\phi_k^{tur} \phi^{turT}_k]. \tag{3.3.26}$$

The temporal correlations of turbulent phase are approximated using an Auto Regressive model of order one (AR1) that provides a linear function to express the evolution of turbulence, based on the Taylor's frozen flow hypothesis [67].

$$\phi_{k+1}^{tur} = A\phi_k^{tur} + v_k \tag{3.3.27}$$

where $v_k$ is a zero-mean white Gaussian noise vector with covariance matrix $\Sigma_v$. The spatial and temporal correlation matrices are related, from (3.3.27):

$$\Sigma_\phi = A^T \Sigma_\phi A + \Sigma_v, \tag{3.3.28}$$

and $\Sigma_v$ is determined in order to preserve the energy of the turbulence

$$\Sigma_v = \Sigma_\phi - A^T \Sigma_\phi A \tag{3.3.29}$$

#### 3.3.1.2 The State-Space Model

For a dynamically evolving system, the state-space model describes the dynamic behaviour of its inputs and outputs by relating consecutive evolution states with first order differential "state" equations. The adaptive optics system can then be expressed with a discrete state-space representation based on the models in the

previous section. Following [3] the state equations are formed:

$$\mathbf{X_{k+1}} = \mathbf{A}\mathbf{X_k} + \mathbf{B}\mathbf{u_k} + \mathbf{v_k} \tag{3.3.30}$$

$$\mathbf{y_k} = \mathbf{C}\mathbf{X_k} + \mathbf{w_k} \tag{3.3.31}$$

where $X_n$ is the state vector that can be chosen to contain the atmospheric turbulence $\phi_k^{tur}$ phase values and mirror command $\mathbf{u_k}$ either in zonal or modal basis, at two consecutive time steps. The mirror command $\mathbf{u_k}$ is the input or control vector, and the WFS measurement $\mathbf{y_k}$ is the output vector in this model. Vectors $\mathbf{v_k}, \mathbf{w_k}$ are gaussian noise vectors with covariance matrices $\Sigma_v$, $\Sigma_w$. The state vector $\mathbf{X_k}$ is defined according to [52]:

$$\mathbf{X_k} = \begin{bmatrix} \phi_k^{tur} \\ \phi_{k-1}^{tur} \\ \mathbf{u_{k-1}} \\ \mathbf{u_{k-2}} \end{bmatrix} \tag{3.3.32}$$

so that *a*) it contains all the required static and temporal knowledge of the system variables, and *b*) the optimal control $\mathbf{u_k}$ is a function of $\mathbf{X_k}$ only.

Matrix $\mathbf{A}$ contains the temporal correlation matrix of (3.3.27), $\mathbf{B}$ holds the

control vector in the state vector and $\mathbf{C}$ follows from (3.3.18) - (3.3.20):

$$\mathbf{A} = \begin{bmatrix} A & 0 & 0 & 0 \\ \mathbf{I} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{I} & 0 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ \mathbf{I} \\ 0 \end{bmatrix}, \qquad \mathbf{C} = \begin{bmatrix} 0 & \mathbf{D} & 0 & -\mathbf{DN} \end{bmatrix} \tag{3.3.33}$$

#### 3.3.1.3 Solution to the Control Problem

Based on this state-space model, the optimal control is obtained in a two step process, by solving the minimum variance estimation problem using a Kalman filter, and a deterministic control problem with a least-squares projection of the estimate.

The Kalman filter predicts the minimum variance estimate in the state vector as the conditional expectation of the phase knowing the measurements in previous time steps. It is generally described by the update and prediction equations

$$\hat{\mathbf{X}}_{\mathbf{k|k}} = \hat{\mathbf{X}}_{\mathbf{k|k-1}} + \mathbf{H}_\infty(\mathbf{y_k} - \mathbf{C}\hat{\mathbf{X}}_{\mathbf{k|k-1}})\hat{\mathbf{X}}_{\mathbf{k+1|k}} = \mathbf{A}\hat{\mathbf{X}}_{\mathbf{k|k}} + \mathbf{Bu_k} \tag{3.3.34}$$

where $\mathbf{H}_\infty$ is the asymptotic Kalman gain, which takes the form

$$\mathbf{H}_\infty = \mathbf{\Sigma}_\infty \mathbf{C}^\mathbf{T}(\mathbf{C}\mathbf{\Sigma}_\infty\mathbf{C}^\mathbf{T} + \Sigma_w)^{-1} \tag{3.3.35}$$

and $\mathbf{\Sigma}_\infty$ is the covariance matrix of the state vector, found by the solution of the Riccati equation which can be pre-computed:

$$\mathbf{\Sigma}_\infty = \mathbf{A}\mathbf{\Sigma}_\infty\mathbf{A}^\mathbf{T} + \Sigma_v - \mathbf{A}\mathbf{\Sigma}_\infty\mathbf{C}^\mathbf{T}(\mathbf{C}\mathbf{\Sigma}_\infty\mathbf{C}^\mathbf{T} + \Sigma_w)^{-1}\mathbf{C}\mathbf{\Sigma}_\infty\mathbf{A}^\mathbf{T} \tag{3.3.36}$$

Once the estimate is predicted from the Kalman filter, the deterministic control problem that computes the control vector $u_k$ is solved by projecting the estimate to the DMs, in equivalence with (3.3.22) - (3.3.23).

$$\mathbf{u_k} = \mathbf{P}\hat{\phi}^{tur}_{k+1|k} \quad , \tag{3.3.37}$$

and in terms of the state vector, it becomes

$$\mathbf{u_k} = \mathbf{K}\hat{\mathbf{X}}_{\mathbf{k+1|k}}, \qquad \mathbf{K} = \begin{bmatrix} \mathbf{P} & 0 & 0 & 0 \end{bmatrix} \tag{3.3.38}$$

### 3.3.2 The Ensemble Kalman Filter

The Kalman filtering process described in the previous section requires a very large number of variables that depends on the number of states used. It is thus becoming computational demanding, and one of the problems is the computation and storage of the estimation error covariance matrices. The Ensemble Kalman Filter (EnKF) approach for adaptive optics proposed in [78] uses an approximation of the original Kalman Filter, based on a Monte-Carlo samples "ensemble" in place of the exact estimation error covariance matrices, avoiding this way explicit computation of the Kalman gain via the Riccati equations.

More specifically, an initial ensemble of $m$ elements is created, that are much less than the actual number of samples in the problem, from the spatial covariance matrix using the Von Karman spectrum of turbulence. This ensemble is used for the system state propagation. Then, at each step, the state covariance matrices are replaced by those computed by the ensemble elements. During the prediction step,

each ensemble element also propagates with the state equation:

$$\hat{\mathbf{X}}^{\mathbf{i}}_{\mathbf{k|k-1}} = \mathbf{A}\hat{\mathbf{X}}^{\mathbf{i}}_{\mathbf{k-1|k-1}} + \mathbf{v}^{\mathbf{i}}_{\mathbf{k}} \tag{3.3.39}$$

The state error covariance matrix is then computed, to be used instead of $\Sigma_\infty$:

$$\Sigma_{k|k-1} = \frac{1}{m-1}\sum_{i=1}^{m}(\hat{\mathbf{X}}^{\mathbf{i}}_{\mathbf{k|k-1}} - \bar{x}_{k|k-1})(\hat{\mathbf{X}}^{\mathbf{i}}_{\mathbf{k|k-1}} - \bar{x}_{k|k-1})^T,$$
$$\bar{x}_{k|k-1} = \frac{1}{m}\sum_{i=1}^{m}\hat{\mathbf{X}}^{\mathbf{i}}_{\mathbf{k|k-1}} \tag{3.3.40}$$

At the correction step, using the previous state ensemble, the ensemble elements are updated with the estimation equation:

$$\hat{\mathbf{X}}^{\mathbf{i}}_{\mathbf{k|k}} = \hat{\mathbf{X}}^{\mathbf{i}}_{\mathbf{k|k-1}} + \mathbf{H_k}(\mathbf{y_k} + \mathbf{w}^{\mathbf{i}}_{\mathbf{k}} - \hat{\mathbf{y}}^{\mathbf{i}}_{\mathbf{k|k-1}}) \quad , \tag{3.3.41}$$

and the Kalman gain $\mathbf{H_k}$ is computed at each time step $k$ from the ensemble-created covariance matrices. The optimality of this method is depended on the ensemble size, so this needs to be large enough to accommodate optimal results. According to [78] the computational complexity of the operations is linear with the order of the system and the number of states, since the computationally heavy gain calculations operate on the ensemble matrices, which are much smaller than the order of the system.

Extensions of this technique for AO control are the Ensemble Transform Kalman Filter (ETKF) proposed in [80], and an improvement that also reduces the computational cost, the Local Ensemble Transform Kalman Filter (LETKF) [79, 81]. In

particular the LETKF employs localized updates to deal with the problem of underestimation of the true covariance matrices due to small state error representations as well as large spatial distances in elements caused by a small ensemble size. This localisation is achieved using domain decomposition methods which are independent and can be done in parallel, thus increasing execution speed.

## 3.4 Other Methods

### 3.4.1 Cumulative Reconstructor with Domain decomposition

Zhariy et al. first proposed the Cumulative Reconstructor (CuRe) in 2011 [82], which is a direct reconstruction method with computational complexity $\mathcal{O}(n)$. The algorithm has been improved to adapt to telescope apertures and to use the Fried sensor geometry [83], and later a domain decomposition method was incorporated to deal with high noise propagation on large apertures, to become the Cumulative Reconstructor with Domain decomposition (CuReD) method [62]. The reconstructor has been originally developed for single-conjugate AO systems, but it can be used as a component in atmospheric tomography as well [35]. Numerical results have shown the highest computational performance for a reconstructor in single conjugate adaptive optics [62, 83], and it has also been demonstrated on-sky with the CANARY MOAO pathfinder [84]. More recently, the method was further improved by U. Bitenc [85] to eliminate a waffle behaviour in the reconstructed image.

#### 3.4.1.1 Cumulative Algorithm

Beginning from the mathematical model for the Shack-Hartmann wavefront sensor, the authors of [82] define the wavefront $\phi(x, y)$ on a quadratic domain $\Omega_T$ that represents the area of the telescope aperture, and subdomains $\Omega_{ij}$ for the subapertures. Denoting with $\phi_x, \phi_y$ the partial derivatives, and assuming that sensor measurements are the averaged phase gradients over subapertures the following relationship is defined for the SH sensor operator $\Gamma = [\Gamma_x, \Gamma_y]$

$$s_x[i,j] = \Gamma_x \phi[i,j] := \frac{1}{|\Omega_{ij}|} \int_{\Omega_{ij}} \phi_x(x,y) d(x,y) \tag{3.4.42}$$

CuRe applies an integration method using the sensor measurements on the $x$ and $y$ direction of the subapertures, to build chains of concatenated measurements. It starts by initially defining the wavefront points at the mid-points of the subaperture sides, according to the modified Hudgin geometry [86]. The relationship between sensor measurements and wavefront values becomes

$$s_x[i,j] \approx \phi\left[i, j - \frac{1}{2}\right] - \phi\left[i - 1, j - \frac{1}{2}\right], \tag{3.4.43}$$

$$s_y[i,j] \approx \phi\left[i - \frac{1}{2}, j\right] - \phi\left[i - \frac{1}{2}, j - 1\right]. \tag{3.4.44}$$

The chains are built from a starting value 0 by adding the wavefront measurement of the current subaperture to the previous point in the chain in each direction, in

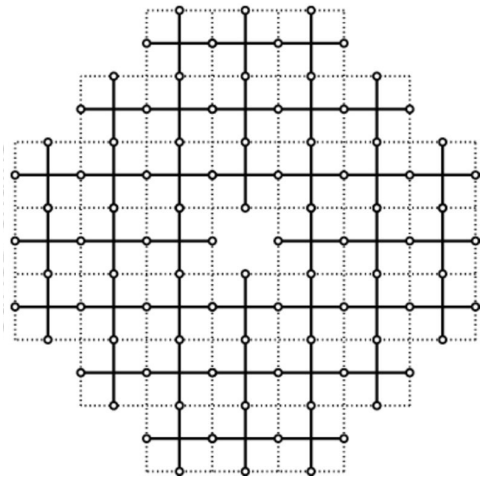


Figure 3.3: Gradient summation chains for CURE (U. Bitenc et al. 2015 [85])

an iterative scheme

$$l_x\left[i+1, j-\frac{1}{2}\right] = l_x\left[i, j-\frac{1}{2}\right] + s_x[i+1, j] \tag{3.4.45}$$

$$l_y\left[i-\frac{1}{2}, j+1\right] = l_y\left[i-\frac{1}{2}, j\right] + s_y[i, j+1] \tag{3.4.46}$$

In order to align the chains between them, they are shifted in each direction by their mean value, so that the mean value of each is 0. If $m_x, m_y$ is the mean value of the $x$ and $y$ chains, they become

$$l_{x0}\left[i, j-\frac{1}{2}\right] = l_x\left[i, j-\frac{1}{2}\right] - m_x\left[j-\frac{1}{2}\right] \tag{3.4.47}$$

$$l_{y0}\left[i-\frac{1}{2}, j\right] = l_y\left[i-\frac{1}{2}, j\right] - m_y\left[i-\frac{1}{2}\right] \tag{3.4.48}$$

Next, trend lines are used to connect the chains, computed from the mean of the

chains in one direction

$$t_x[i] = \frac{1}{N}\sum_{j=1}^{N} l_{x0}[j, j], \tag{3.4.49}$$

and the intermediate points at the crossings of the lines of different directions are found:

$$t_x\left[i + \frac{1}{2}\right] = \frac{1}{2}(t_x[i] + t_x[i+1]). \tag{3.4.50}$$

These points are the mean values of the corresponding chains that provide a correct alignment. The chains are shifted in a way that the mean values in the shifted chain are matched to these points: the $x$ chain is matched to $y$-trend line points and vice versa. At this point, the wavefront estimates are found:

$$\phi_x\left[i, j - \frac{1}{2}\right] = l_{x0}\left[i, j - \frac{1}{2}\right] + t_y\left[j - \frac{1}{2}\right] \tag{3.4.51}$$

Finally, the result needs to be adapted to the Fried geometry:

$$\phi[i, j] = \frac{1}{4}(\phi_1[i, j] + \phi_2[i, j] + \phi_3[i, j] + \phi_4[i, j]), \tag{3.4.52}$$

where $\phi_i[i, j]$ are extrapolations from their 4 neighbouring side points taking into account the subaperture measurement, for example:

$$\phi_1[i, j] := \phi\left[i - \frac{1}{2}, j\right] + \frac{1}{4}(s_x[i, j] + s_x[i, j+1]) \tag{3.4.53}$$

It should be noted that, for circular or annular apertures, the final derivation of the side points cannot be done with this trend line computation method. The trend lines must take into account the number of valid measurements in each line between

two chains when computing the mean, and to modify each iteration for obtaining the intermediate points such that they too take into consideration the changing number of measurements between chains.

#### 3.4.1.2 Domain Decomposition

Domain Decomposition for CuRe is intended to compensate for the high noise propagation that the algorithm exhibits. It is based on the idea that noise propagation increases with the domain size - and thus the chain length - due to a random walk effect, and so, if the domain size where CuRe is acting is restricted, so will the noise be. The telescope aperture domain $\Omega_T$ is decomposed into subdomains $\Omega_{p_i}$, whose intersections only contain their common boundaries.

$$\Omega_T = \bigcup_i \Omega_{p_i} \tag{3.4.54}$$

The domains should not overlap subapertures. Using a hierarchical algorithm, each domain, starting from $\Omega_T$, is divided into four parts, as in figure 3.4.

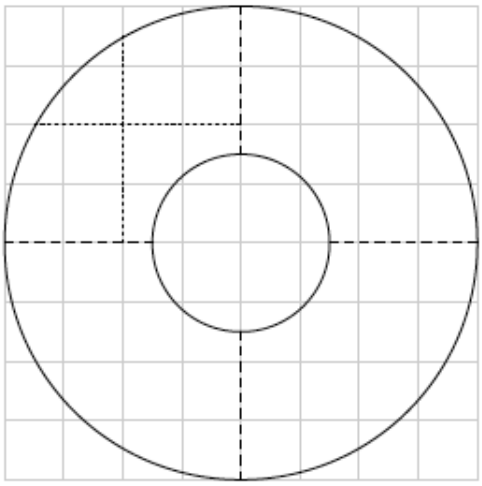


Figure 3.4: Domain decomposition in CuReD (M. Rosensteiner 2012 ).

The CuRe is applied to each subdomain independently, but the solutions sets must then be connected to obtain the wavefront map, which is also done hierarchically following the reverse path of the division. Considering the Fried geometry, wavefront values are located at the boundaries between subdomains, and they can be used to adjust the mean for each reconnected chain to 0, by applying the necessary shift. When 4 parts are fitted together, the mean value differences are computed at the common boundaries as below:

$$d_{1,2} = \text{mean}(\phi_1|_{\partial\Omega_{p_1}\cap\partial\Omega_{p_2}} - \phi_2|_{\partial\Omega_{p_1}\cap\partial\Omega_{p_2}}), \tag{3.4.55}$$

where $\phi_1, \phi_2$ are the phase values from boundaries $\partial\Omega_{p_1}$ and $\partial\Omega_{p_2}$ respectively, at their intersection (denoted by $\cap$). The parts are shifted according to these mean value differences, starting with the second part:

$$\bar{\phi_2} = \phi_2 + d_{1,2}, \tag{3.4.56}$$

$$\bar{\phi_3} = \phi_3 + d_{1,3}, \tag{3.4.57}$$

$$\bar{\phi_4} = \phi_4 + d_{1,2} + d_{2,4}. \tag{3.4.58}$$

After the reconnection of the subdomains is finished, the full wavefront set is shifted to zero mean.

### 3.4.2 Kaczmarz Iteration

This method can be used as an extension of the single-guide-star system to atmospheric tomography with multiple guide stars. It has been presented in [35, 87]

from the developers of the CuReD algorithm. It assumes as input to the tomography problem the already reconstructed phases from each measured direction, which are assumed to have been computed fast, for example using CuReD. Given these estimates, the phases at each turbulent layer are projections of a union of shifted reconstructed areas at the specified layer height, which overlap.

Based on this model, reconstructing the turbulent layers from aperture-plane partial area reconstructed phases consists of solving a system of equations that maps this geometric shift and area combination, that is described by a set of equations:

$$\mathbf{A}_{\alpha_\mathbf{g}}\mathbf{\Phi} = \phi_{\alpha_g}, \qquad g = 1, \cdots, G., \tag{3.4.59}$$

with $G$ number of guide stars, $\Phi$ a vector of unknowns of the concatenated layer phase values, $\alpha_g$ denoting guide star angle, $\phi_{\alpha_g}$ the known aperture-plane reconstructed phases for a specified guide star $g$, and $\mathbf{A}_{\alpha_\mathbf{g}}$ the per guide star layer-to-aperture projection operator. The Kaczmarz method acts on each layer equation separately, using the following iterative scheme:

Table 3.1: Kaczmarz Iteration

Chose $\mathbf{\Phi_0}$
**for** $i = 1, \cdots$ **do**
  $\mathbf{\Phi}_{i,0} = \mathbf{\Phi}_{i-1}$
  **for** $g = 1 \cdots, G$ **do**
    $\mathbf{\Phi_{i,g}} = \mathbf{\Phi}_{i,g-1} + \beta_g \mathbf{A}^*_{\mathbf{ag}} \mathbf{r}(\mathbf{A_{ag}} \mathbf{A}^*_{\mathbf{a}} \mathbf{g})(\phi_{ag} - \mathbf{A_{ag}} \mathbf{\Phi}_{i,g-1})$
  **end for**
  $\mathbf{\Phi}_i = \mathbf{\Phi}_{i,G}$
**end for**

In this algorithm, $\beta_g$ is a scaling parameter and $r$ is an operation-valued function, such as to fulfil:

$$r(\mathbf{A_{ag}} \mathbf{A}^*_{\mathbf{ag}}) = (\mathbf{A_{ag}} \mathbf{A}^*_{\mathbf{ag}})^{-\mathbf{1}}. \tag{3.4.60}$$

The operators $\mathbf{A_{ag}}$ and $\mathbf{A_{ag}^*}$ can be evaluated in the iterations without the need to use matrix-vector operations, which makes the process numerically cheap. It is also shown in [35] that the algorithm converges geometrically to a least-squares solution, and that noise regularization is achieved by terminating iterations early. The number of numerical operations required per iteration is $G \times (18 \times L - 14) \times N$ and convergence is achieved with 2 - 3 iterations. Additional work in [87] also treats the LGS effects.

## 3.5 Discussion

### 3.5.1 Summary of Computational Performances

Table 3.2 summarises the computational costs of the algorithms presented above. The reconstruction total problem size is denoted with $N$, which applies to both single guide star AO and tomography. A tomography measurement vector increases in size with the number of guide stars, while the phase vector increases with the number of corrected atmospheric layers, so the two vectors may differ multiple times in size. Hence we take $N$ to refer to the largest vector. Where a sparse operation is involved, $nnz$ is the number of non-zero elements per row or column. The computational complexity, expressed with the Big O notation ($\mathcal{O}$) provides an indication of the increase in execution time as a function of the problem size. For the EnKF, the problem size is defined by the ensemble elements $m$, as defined in 3.3.2. Note that $m$ depends on the required accuracy of the result, and it can be close to $N$. Actual execution times are not reported here, as the AO system configuration parameters

that were used in the tests of reported results for each algorithm vary, and they do not provide a meaningful measure for comparison. Execution time results can be found in the references of each respective section in this chapter, in AO systems design documents for the E-ELT [73] and TMT [88, 89], as well as in chapter 6 of this thesis.

Table 3.2: Summary of Computational Complexities

| **Algorithm** | **Memory** | **# operations in real-time** | |
|---|---|---|---|
| | | cost per iteration | # iterations |
| MVM | $> 5$ GB | $\mathcal{O}(N^2)$ | 1 |
| CG | $< 50$ MB | $\mathcal{O}(nnzN)$ | 30 |
| FDPCG | $< 50$ MB | $\mathcal{O}(N \log(N)$ | 3 |
| FrIM | $< 50$ MB | $\mathcal{O}(N)$ | 3 |
| CuReD | | $\mathcal{O}(N)$ | 1 |
| LQG | | $\mathcal{O}(Nnnz)$ | 1 |
| LETKF | | $\mathcal{O}(m^3)$ | 1 |
| Kaczmarz | | $\mathcal{O}(N)$ | 2-3 |

### 3.5.2 Suitability and Potential for Parallel Processing

From the previous descriptions, we can identify the algorithms that are more suitable for parallel execution, and those that can benefit the most from a parallel implementation in the context of their target AO architectures.

The arithmetic complexities of table 3.2 are only one measure of the computational burden involved in the reconstruction calculations, they do not reflect, however, the time in which they can be achieved, because this varies according to the suitability for, and the level of, parallelism that can be applied to each algorithm,

as well as the features of the parallel hardware that will be used. Starting from the conceptual characteristics, we can see immediately that there is an obvious potential for parallel operations where there are independent parts to be calculated. This is evident in the case of Matrix Vector Multiplication (MVM), that can be directly divided into smaller matrix-vector problems, thus able to provide a linear scaling in speed to the number of sub-problems. CuReD, and LETKF that work separately on sub-domains, with some overhead involved in recombining the full domain also indicate a good potential. The Kaczmarz iterations - although iterative - can operate on each atmospheric layer separately, thus layers can be solved in parallel. For CG, FDPCG and FrIM, this is not straight-forward, as they are recursive methods that require a set number of iterations to complete, in which case, from an implementation point of view, one has to look at how fast each iteration can be computed, and look for parallelism opportunities within this scope. In that respect, FDPCG incorporates grid-based block matrix structures that can be operated on in parallel. The description of FrIM does not immediately imply parallel computational regions, but methods for applying parallelism which showed vast improvement have been demonstrated in [73]. The main cost for applying CG is a sparse MVM, that can also be done in parallel like the dense  using high performance libraries.

Another thing to consider is the limiting factor in each algorithm, with respect to the percentage of the theoretical speed the execution can reach based on the number of computations, when limited by another aspect of hardware latencies. A significant factor is usually the memory bandwidth related to the number and pattern of memory transfers / accesses. Another is the synchronisation costs for

the parallel scheme employed and device used. The performance bound due to bandwidth can be expressed [90]:

$$\text{flop rate} = \frac{\text{flops in algorithm}}{\text{bandwidth required (bytes)}} \times \text{available bandwidth.} \tag{3.5.61}$$

As an example, the MVM is a bandwidth-dependent operation. There are not enough computations compared to the memory transfers needed. The flops required are $2n^2$ and the bytes to be transferred $\sim 2n^2 \times 4$, so from (3.5.61) we conclude that the performance scales with about a quarter of the device memory bandwidth, and that the percentage of peak performance achieved depends on the ratio of available bandwidth to bandwidth required:

$$\text{max performance}(\%) = \frac{\text{available bandwidth}}{\text{bandwidth required}} \times 100 \tag{3.5.62}$$

Thus, in theory, the operation is completely bandwidth-depended. In practice, the maximum amount of available memory relative to the problem size also poses a cap in the maximum performance achieved. That is, if the hardware used can only achieve memory throughput after a certain vector size, as is the case for GPUs, smaller vectors will have significantly less, and constant performance. Of the rest of the algorithms, those that perform a matrix-vector operation explicitly, will also suffer from the same limitations.

Iterative algorithms suffer also from synchronisation costs at every iteration when

implemented in parallel, thus to accelerate execution by using parallelism, the gain from applying it should greatly exceed these costs to provide significant speedup. This has reportedly been a problem for example with FrIM [73]. Additionally, if the problem were to be divided between multiple devices such as multiple CPU nodes or GPUs there would be further synchronisation costs. However, CG, FrIM and FDPCG all have small memory requirements and this shouldn't be the case. Therefore the question is how much overall speedup can be achieved in a single device when using multicore / manycore architectures.

CuReD has shown an impressive computational performance by the use of subdomains in a CPU parallel thread implementation, and already seems to greatly exceed the real-time requirements in the single WFS AO case. However, if used for tomography with Kaczmarz iterations, which can also be run in parallel, it makes sense to assess the full computational ability of the process on GPUs. FrIM, and LETKF also use domain methods, and they have a good potential for further parallelism gains, however, their application to tomography is still immature (for FrIM-3D) or not developed yet (for LETKF). The LQG method is an optimal solver, that has given very good results so far in terms of AO tomography performance, but it requires a relatively larger amount of computations in real time, the dominant cost being a matrix vector multiplication with the associated problems, and although the reduction of the otherwise huge cost of off-line computations, which is also important, has being studied at some extent, on line computations haven't been explicitly considered yet. The off-line computation of the Kalman gain matrix has been made parallel, by using a distributed control approach [53], but it is possible that there is

room for optimization.

#### 3.5.2.1 Parallel Processing Hardware

Currently the main options in parallel hardware are multicore CPUs, manycore co-processor platforms, and reconfigurable devices. Multi-core CPUs to date can provide a limited number of parallel threads when working with very large datasets. Manycore platforms like the GPU and Intel Xeon Phi, have up to 2880 and 61 floating point cores respectively. Field Programmable Gate Arrays (FPGAs) are low powered devices which are also capable of performing well in massively parallel applications, but they are less flexible in programming effort. The most favourable choice currently is NVIDIA GPUs because they have the most resources available both for computing and memory storage, their ease of use, compatibility with other software and consistency over the years, and the fact that they so far seem to perform much better in many aspects in high performance scientific applications [91]. This thesis will concentrate on the use of GPUs in accelerating wavefront reconstruction algorithms to an E-ELT scale, and details of the device architecture will be given in chapter 5.

The key to exploiting the computational power of any parallel device is to adapt the application to make optimal use of the hardware resources available. In doing this, the first step is to identify where there are opportunities in the algorithm for data-parallel processing. In the case of wavefront reconstruction, dense matrix-vector multiplications are highly data parallel operations, since they can be applied to different data independently and the direct inversion required can also be done in parallel, both with the use of standard libraries on CPUs and GPUs, and with well

studied schemes on FPGAs. This approach has the disadvantage of the requirement for storage of the large reconstruction matrix, which will also need to be updated at regular intervals - although it is now possible to store it on as little as 1 GPU [92, 93]. In addition, matrix-vector multiplication is inherently a bandwidth limited operation, meaning that the maximum performance that can be obtained from a single device is limited by the transfer bandwidth between the processor and its main memory [90]. It is however the simplest method requiring less programming effort with standard libraries available for the implementation, so in this respect, it makes sense to assess the maximum performance achievable on parallel hardware. Sparse direct methods such as the Cholesky solver can improve this approach, provided the formulations can be adapted to the chosen wavefront reconstruction model [49].

When looking at iterative methods, their application in parallel is not so straight forward, partly because of the recursive nature of these algorithms. If data is to be broken down in parts for parallel accesses, they have to be recombined and synchronised before the next iteration starts, to avoid race conditions, and this can potentially be a bottleneck for performance. The operations involved in the PCG method are also even more bandwidth limited and optimisations in data layout, caching and access patterns is essential. Advantages of this method for hardware acceleration are that since all operators are sparse, with the increased storage capacity of GPUs it is possible that the problem can be solved using only 1 device and thus multiple-device level synchronisation can be avoided. Recent advances on GPU technology offer features for using very efficient libraries for the sparse computations within the same device program, thus avoiding various overheads and data transfers

associated with library function calls. The proposed algorithms themselves already contain features to optimise memory consumption and access latency / bandwidth usage (see for example the FrIM algorithm [49]), that can also be exploited advantageously on GPUs.

# Chapter 4

# Graphical Processors for Real-time Control

In this chapter, I review the history of graphical processors as general purpose programmable devices, and describe their architecture and use. I present the function and limitations of a real-time control pipeline for AO and discuss how the features of GPU devices fit these limitations.

## 4.1 Graphical Processing Units

The Graphical Processing Unit (GPU) is currently one of the candidate platforms for performing Real Time Control System (RTCS) tasks for the future AO systems on ELTs. It is favoured today by many scientists and professionals that use high performance computing for their applications. There are three main reasons for this: First, the potential for a large increase in application speed, second, the compatibility and ease of use with general purpose computing standards, which requires less

programming effort, and third, the fact that they are relatively low-cost off-the-shelf products, which reduces the hardware and maintenance costs and effort.

GPUs have traditionally been implemented as hardware accelerators to handle calculations for computer graphics-specific tasks. With the emergence of three-dimensional graphics technologies in the 90's there was an increased interest in using these devices for general purpose computations, especially as they demonstrated very high computational power owing to their parallel architecture. By the beginning of the last decade, the continuous effort in the field of 3D graphics, had resulted in a turn in the design approach of graphics applications, from configuring fixed modes of operations for data elements in traditional graphics card processing, towards programming parts of the graphics pipeline, a feature called programmable "shading". Shading allowed the capability of applying mathematical operations to data that represent image features (vertices) by means of a short program running on the GPU. This ability to program the graphical processor, gave rise to a new research area that studied General-Purpose computation on Graphics Processing Units (GPGPU) [94], that has had vast activities in the last 13 years, making big contributions to numerical and scientific programming. Initially, programmers made use of the devices by mapping scientific algorithms to the existing graphics programming specific Application Programming Interfaces (APIs), and later, the BrookGPU project from Stanford University and the Sh project from the University of Waterloo, provided extensions to high level programming languages (see [94] for relevant information and resources). As of 2006, NVIDIA and later ATI have both launched a new type of architecture that exploits the similarity of the GPU pipeline to the

stream processor model [95, 96].

The remainder of this section will be a description of the currently most prevalent of these two architectures, the NVIDIA CUDA platform [97]. There is extensive literature on the history, architecture and applications of the GPU, on which much of the material in this section is based, and additional resources can be found in [94, 97, 98].

### 4.1.1 Hardware Architecture

The architecture of the modern GPU has evolved in the framework of graphics rendering, thus with the objective of performing operations with high arithmetic intensity in parallel to cope with the amount of data and calculations required for 3D imaging. This high degree of parallelism is accomplished by dedicating more of the hardware resources on the chip to data processing, and less to control blocks and cache, as opposed to a CPU, as shown in figure 4.1 [99]. The GPU operates as a co-processor in a host system.

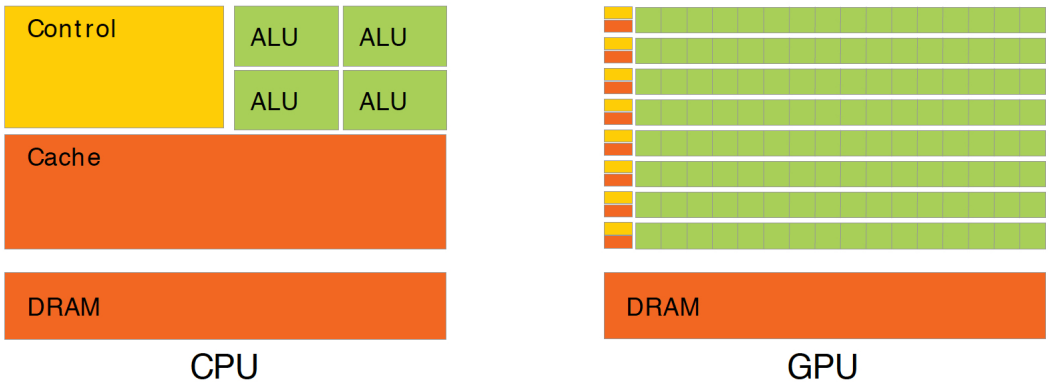



Figure 4.1: CPU and GPU allocation of resources on chip (NVIDIA [99])

#### 4.1.1.1 The Classic GPU Pipeline

The GPU hardware has always been tailored for graphics applications, and graphics technology terms are used to describe it. The data input from the host are thought to represent vertices with various attributes. A set of dedicated processing units called the vertex processor, apply hardware shading functions or shader programs, and next, the data are assembled to basic geometrical units and converted to pixel fragments. From there, in the same fashion, the fragment processor applies the final functions that produce the complete pixels. A number of rasterizing operations then finalize the three-dimensional image for output to the display. This basic sequence, shown in figure 4.2 comprised the classical graphics pipeline for the 20 years prior to the "unified" architecture [99].

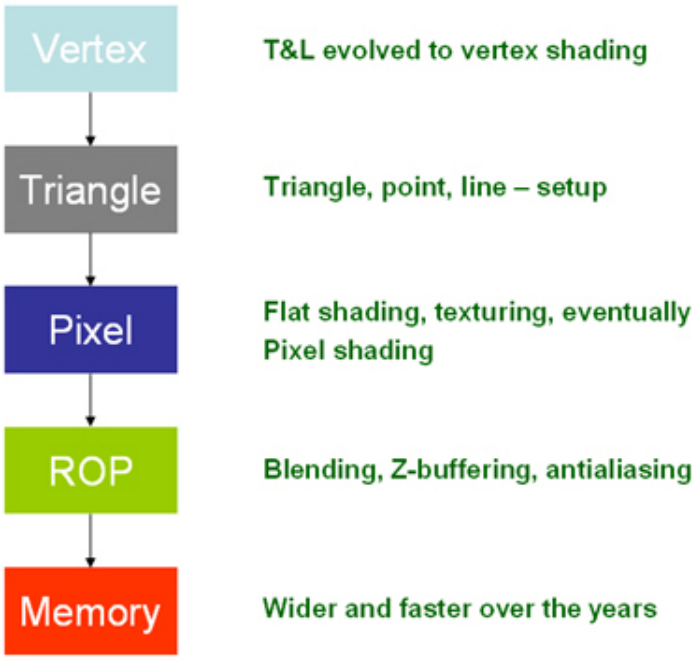



Figure 4.2: The classic GPU pipeline: Processor per-function (NVIDIA [99])

Classic pipelines had a number of significant limitations in manipulating data. Incomplete floating-point implementation, no integer arithmetics, restrictions on

reading and writing memory throughout the pipeline, and limited resources in terms of memory structures, shader instructions etc., are some of the most relevant to general purpose computing as well as to graphics processing. These, along with the drive to be compatible with Microsoft DirectX 10 graphical applications [100], were some of the things that led to the next major change in architecture.

#### 4.1.1.2 Unified Shader Architecture

Older GPUs employed dedicated vector processors to manipulate the different main processing stages of the pipeline. With the introduction of the nvidia G80 processor in 2006 [99], the graphics pipeline now processed all data using a unified shader core (in practice a set of scalar floating point processing units) with all its computing units available for all types of operations. The processing cycle takes place in loops, reducing the sequential length and hardware complexity. This unification increases performance by maximizing the use of computing units available on the device that might otherwise have a significant idle part if fixed numbers of dedicated units were used. It provided a peak floating-point performance of up to 500 Gflops. The simplified hardware model of the device oriented to general purpose computing was implemented as a set of Single Instruction, Multiple Data (SIMD) "multiprocessors" with on-chip memory (figure 4.4). Each multiprocessor in this first generation of the Unified Architecture has 8 stream processors that execute the same instruction simultaneously on different data, making a total of up to 128 processors on a single chip (Geforce 8800GTX) [99]. Each stream processor is a scalar processor operating on a single data element, is capable of single precision floating point operations and is clocked at a multiple of the core clock frequency providing high-speed instruc-

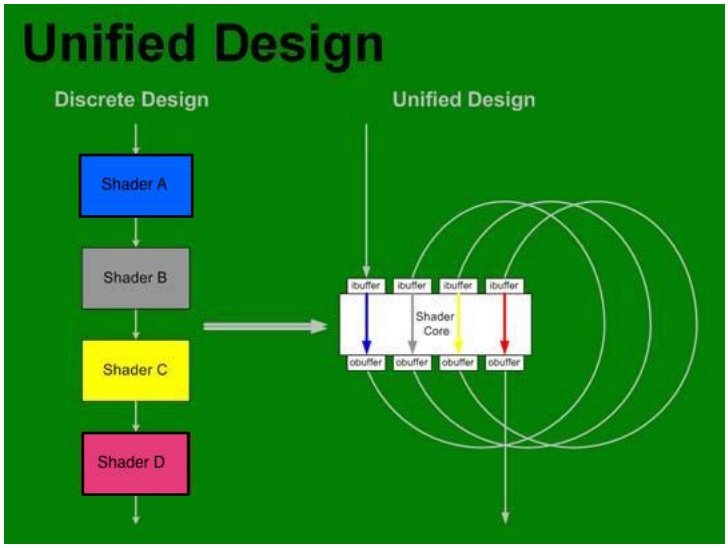



Figure 4.3: Classic v Unified Shader Architecture (NVIDIA [99])

tion execution. Each of the top blocks in figure 4.4 is a multiprocessing unit with dedicated memory, employing several scalar processors and executing instructions in parallel with the others, managed by a thread scheduler. Data is transferred from and to the external memory on the card.

An improvement in G80 came in 2008 with the GT200 processor [101], which marked the second generation of the unified NVIDIA GPU architecture. This featured an increased number of computing cores to 240, doubled register file size, improvements in hardware memory access patterns, and double precision arithmetics support, among others. Performance was nearly doubled.

In later architectures, the SIMD architecture model was replaced with Single Instruction, Multiple Threads (SIMT). SIMT is different to SIMD in that it specifies the execution and behaviour of a single thread, while SIMD is a vector organization with its *width* exposed to the software.

NVIDIA GPU hardware architectures are since then classified with their Compute Capability (CC), which assigns a version number to the hardware, indicating

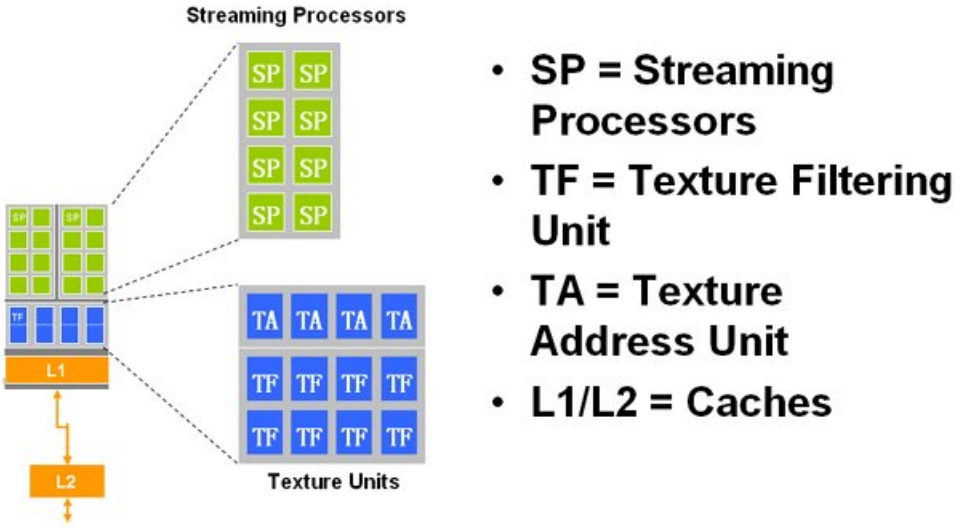



Figure 4.4: G80 Streaming Processor Model(NVIDIA [99])

the parallel computing features of the device. The G80 has CC 1.0, and the latest architecture, GM200, has CC 5.2.

#### 4.1.1.3 Programming model

The device is seen as a set of streaming processors with on-chip memory. The on-chip memory is shared by all the processors and is organized per multiprocessor as a set of registers, a shared memory, and a constant and texture cache that correspond to special spaces in the external memory. The external DRAM contains three spaces: a global memory that is used to store large amounts of data transferred from the host, a constant memory optimized for low cost accesses, and a texture memory, optimized for fast manipulation of two dimensional data structures. The G80 hardware supports multiple threads through a local thread scheduler to each multiprocessor and a global scheduler at the core level. The programming model is shown in figure 4.5. Threads are executed using time slicing. Since an instruction can be executed simultaneously on different data on the stream architecture, groups

of threads issuing the same instruction are time sliced. The programming model groups threads firstly as warps of 32 threads that are executed simultaneously as SIMD threads. Warps are time sliced by the thread scheduler. Threads are arranged and indexed in blocks of up to three dimensions. In the same fashion blocks fill a grid. The grid is carrying out the tasks on all the specified data, with one multiprocessor executing at least one block. This way all threads in a block can have fast memory accesses by sharing the multiprocessors' on-chip memory. The model is scalable: The total number of threads launched at once can be very large to fit to the on-chip resources so some of the thread blocks will be serialized, and the same program is able to execute in devices of different parallel capabilities.

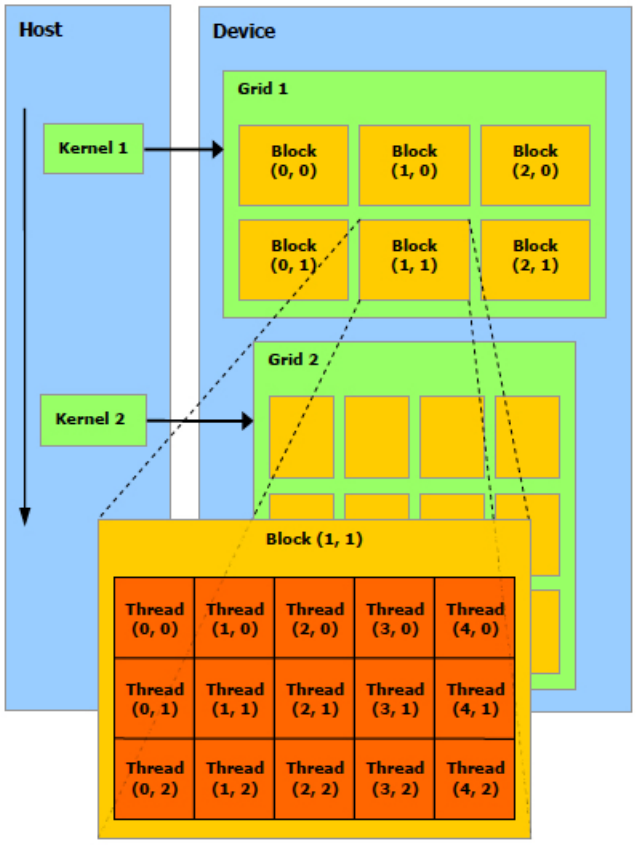



Figure 4.5: CUDA Programming Model(NVIDIA [102])

#### 4.1.1.4 Compute Unified Device Architecture

The NVIDIA Compute Unified Device Architecture (CUDA) emerged in 2006 as a new hardware and software architecture that exposes the device as a programmable stream processor and allows program management in ways similar to the conventional CPU. The CUDA API is based on PTX ISA [103], NVIDIA's low-level Parallel Thread Execution Instruction Set Architecture that exposes the GPU hardware as a data parallel computing device that executes a very large number of parallel threads, operating as a co-processor to a main CPU host machine. CUDA provides a high level programming interface to the PTX model, including a set of language extensions that allow C/C++ and Fortran code to be translated to GPU machine code. A GPU function, called a kernel can be defined and launched using these language extensions. Kernel functions are asynchronous with respect to the host, and asynchronous host - device memory copies are also implemented. Devices are capable of performing memory copies concurrently with kernel executions, a functionality that is handled with "streams", that are objects that get assigned a sequence of separate operations, and execute asynchronously with respect to each other, in a way similar to CPU threads.

The first CUDA software provided a runtime library that allows manipulation of the device resources and use of mathematical functions, a driver interface as well as two high level libraries, a Basic Linear Algebra Subroutines (BLAS) [104] implementation, cuBLAS, and a Fast Fourier Transforms library, cuFFT. Gradually, support for numerical libraries has grown, both from NVIDIA and community research, and the current version of CUDA, CUDA 7.5, includes with it more libraries. Some

of these are a random number generator library, cuRAND, a sparse matrix BLAS implementation, cuSPARSE, and cuSOLVER, a high level package that provides Linear Algebra PACKage (LAPACK) [105] routines for matrix solvers and factorizations such as SVD, that also supports some types of sparse matrices. In addition the CUDA toolkit provides a graphical Integrated Development Environment (IDE), as well as integration with IDEs such as the Microsoft Visual Studio, and a set of useful tools for debugging, profiling, and other relevant utilities.

### 4.1.2 Current Architectures

An increasing interest from researchers and High Performance Computing (HPC) professionals in GPU development has formed a populous community mainly around the NVIDIA CUDA platform, which gathered momentum such as to rapidly drive in the last two years major advances in the API, along with architectural changes that supported a much improved programming model, that is even closer to the standards of conventional CPU based application development.

A more detailed look at the two last NVIDIA processors generations, codenamed "Fermi" [106] and "Kepler" [107] is presented here in order to describe the current functionality of NVIDIA GPUs.

#### 4.1.2.1 Fermi Architecture and CUDA 4.0

The NVIDIA GF100 architecture [108], codenamed "Fermi" was launched with the release of GeForce 400 series graphics cards in 2010. Fermi's HPC version was available shortly after [109]. CUDA 4.0 came to production several months later to provide the new features for exploiting the additional computational capabilities

of that architecture [110]. It provided a single precision peak performance of 1.5 TFlops, and up to 6GB card memory.

The Fermi chip design has some significant differences from its predecessor, GT200. These are illustrated in figure 4.6, and summarised in the table of figure 4.7. The first and most visible difference is the change in the structure of the Streaming Multiprocessor (SM) itself, which is altered to include 32 cores instead of 8, and double the total number of cores, with the use of 40nm CMOS fabrication process and a total size 42.5 $\times$ 42.5mm. These NVIDIA "Third Generation SM" cores consisted of an Floating Point Unit (FPU) fully compatible with the current floating point standard, and an Arithmetic Logic Unit (ALU) supporting full 32-bit precision integer arithmetics, as opposed to 24-bit in GT200. This is a significant improvement because it eliminates the need for multiple instructions that existed previously to achieve integer multiplications. Another major improvement in the hardware is the addition of a second thread scheduler. This allows double the number of threads to run simultaneously in the same logical block than its predecessor GT200. This also enables the ability to run kernels concurrently in CUDA applications. The on-chip memory subsystem is reconfigured to provide an L1 cache within the shared memory, a further step in the memory hierarchy between local and global memory, that automatically caches data for programs that do not allocate shared memory. The L1/shared memory space size is also significantly increased (see figure 4.7).

In addition, the architecture introduced the support for the second generation PTX ISA that enabled several of the new additions to the programming model,

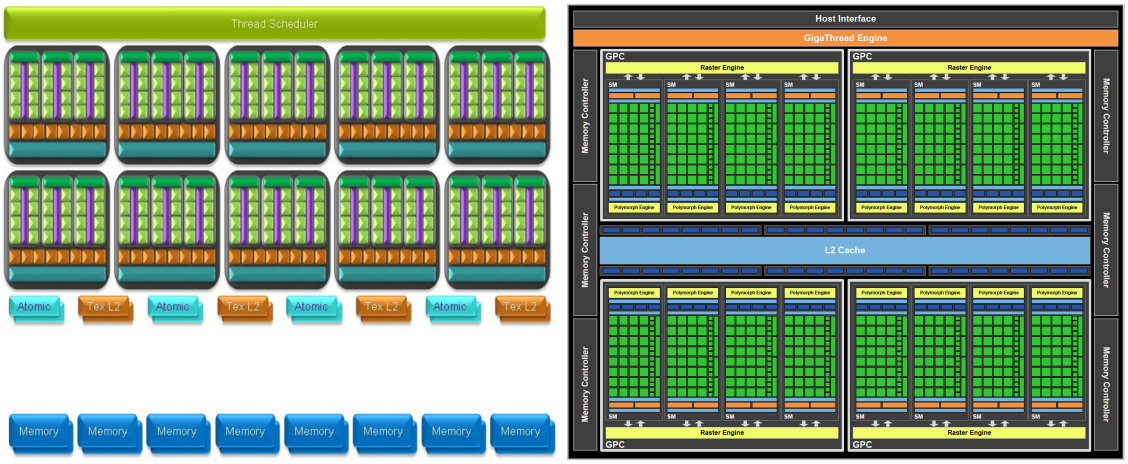



Figure 4.6: Graphical illustration of GT200 (left) and Fermi (right) GPU architectures (NVIDIA [101, 106]).

including 64-bit addressing, full IEEE 32-bit floating point compliance, and a common address space for all variables. This way, it achieved compliance with several compilers, compatibility with different GPU generations, and scalability to GPU core numbers. The ability to use inline PTX instructions provides for fine-grained design of compute applications.

| GPU | G80 | GT200 | Fermi |
|---|---|---|---|
| **Transistors** | 681 million | 1.4 billion | 3.0 billion |
| **CUDA Cores** | 128 | 240 | 512 |
| **Double Precision Floating Point Capability** | None | 30 FMA ops / clock | 256 FMA ops /clock |
| **Single Precision Floating Point Capability** | 128 MAD ops/clock | 240 MAD ops / clock | 512 FMA ops /clock |
| **Special Function Units (SFUs) / SM** | 2 | 2 | 4 |
| **Warp schedulers (per SM)** | 1 | 1 | 2 |
| **Shared Memory (per SM)** | 16 KB | 16 KB | Configurable 48 KB or 16 KB |
| **L1 Cache (per SM)** | None | None | Configurable 16 KB or 48 KB |
| **L2 Cache** | None | None | 768 KB |
| **ECC Memory Support** | No | No | Yes |
| **Concurrent Kernels** | No | No | Up to 16 |
| **Load/Store Address Width** | 32-bit | 32-bit | 64-bit |

Figure 4.7: Comparative summary of features between GPU architectures up to Fermi (NVIDIA [106])

One of the key concepts in the programming model is the Unified Virtual Address (UVA), illustrated in figure 4.8. The introduction of UVA meant that the programmer would now be able to use the full functionality of pointers in the code, also complying with C++. It releases the need for manual address mapping in order for it to be correctly recognized and handled by the three separate load /store instruction sets previously used. A single load / store instruction set is used since the address translation to the different types of memory is handled automatically in the hardware.

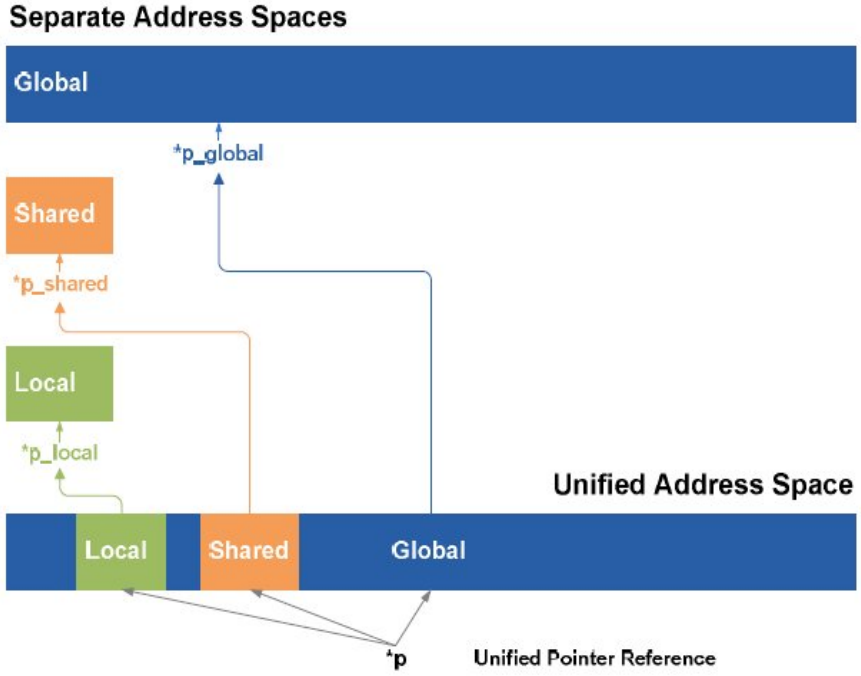



Figure 4.8: Unified Virtual Address space conceptual representation. (NVIDIA [106])

In terms of performance, the highlight is on the increase in speed of computations with double precision numbers, and in memory bandwidth. These are complemented, in line with the multiprocessors structure, with larger, reconfigured local storage and more load / store units per multiprocessor, to facilitate the input of larger chunks of data, which also helps maximize processor occupancy.

CUDA 4.0 was designed to implement the programming model supported by the Fermi architecture, with the aim to take advantage of all the new hardware features. It provided the means to increase performance by integrating in the API all the necessary functionality for fine grained parallelism. The API also contained the tools for interfacing the GPU, for kernel code debugging, and for profiling, in an easy to use set of extensions. The API was restructured to form around the Low Level Virtual Machine (LLVM) compiler which is language independent, has support for a broad range of high level compilers and hardware, and acts as an intermediate stage that translates compiled instructions to machine-depended code [111]. This way it can utilize the architectural innovations on Fermi, but also open-up CUDA to more platforms [110].

#### 4.1.2.2 Kepler Architecture and CUDA 5.0 - 7.0

The next major step in architecture and the one currently used is the GK100 and GK200 series, codenamed "Kepler". Peak performance is increased to over 1 TFlops for double precision, and up to over 4 TFlops for single precision. The changes in hardware include a new multiprocessor design, enhancements in the memory sub-system and increased on-chip and per-thread resources. The instruction set is also upgraded, and in combination with the hardware changes some new programming capabilities are enabled.

The new multiprocessor architecture called SMX (figure 4.9) has 192 single and 64 double precision cores, as well as special function units and load / store units with access to a local 64 KB shared memory / L1 cache. Unlike G80 and Fermi, the cores are clocked at a lower rate, to reduce energy consumption and increase the

ratio of performance/Watt. The SMX employs 4 warp schedulers, allowing more groups of threads to execute concurrently. Each thread has access to 255 registers, alleviating register spilling conditions that may slow down execution. In GK200 (Tesla K80) the register file size and the shared memory / L1 size are doubled. The ability for data sharing between two threads in a warp directly is also added. The texture units are increased in numbers and the limits on usage are lifted. The memory subsystem is similar to Fermi, with an additional local read-only data cache, increased bandwidths, and additional user configurations. The L2 cache is doubled in size and provides up to double bandwidth.


Figure 4.9: The Kepler SMX multiprocessor design. Bright green squares for floating point cores, orange for double precision units,dark green for special function units. (NVIDIA [107])

A new feature called dynamic parallelism is introduced, and it allows device code to be launched from the device. This means that each compute kernel can execute new kernels dynamically. To accommodate this, the work distribution design has changed, with a new Grid Management Unit that is able to prioritise and suspend workload (grids) to be dispatched. Dynamic parallelism can be beneficial for recursive or data dependent operations, and helps with better computing load distribution. This feature enables device executable libraries, such as cuBLAS to be called dynamically from the device. Work scheduling has an additional feature, called Hyper-Q, that allows multiple work queues connections between the host and GPU, improving stream concurrency and performance in applications using multiple host threads / processes to access a GPU.

#### 4.1.2.3 Maxwell and Future Architectures

NVIDIA has released the full Maxwell architecture in September 2014 [112], that introduced some changes over Kepler. The multiprocessor design has changed to the SMM multiprocessor, with changes to the logic partitioning, instruction scheduling and allowing double the number of thread blocks per SM. Notable changes are also the doubling of shared memory per SM, larger L2 cache, and reduced arithmetic instruction latencies. Maxwell has only been developed for gaming cards so far.

The next generation of GPU expected in 2016 which was recently announced is Pascal [113, 114]. It is anticipated that it will feature a new type of memory called 3D or stacked memory that is said to triple the memory bandwidth, and the NVIDIA NVLink High-Speed Interconnect, a new host - GPU communications channel that claims to deliver 5 - 12 times the bandwidth of PCI express (PCIe) v.3.0 [115]. It

will also feature up to 32 GB card memory, and mixed precision computing.

#### 4.1.2.4 cuBLAS and cuSPARSE

The CUDA libraries that are used in this study are cuBLAS and cuSPARSE [116], which are very widely used in linear algebra problems in GPUs.

cuBLAS is a GPU implementation of the BLAS for CUDA. It contains all the basic dense, banded and triangular matrix and vector operations, and is thread safe with respect to the host. The basic workflow when using cuBLAS is to transfer the data to be operated on to the GPU, and launch the linear algebra functions from the host. Asyncronous execution is also supported via streams. As of CUDA 5.0, cuBLAS has added a device library that uses dynamic parallelism to enable launching functions from within the device on GPUs of compute capability 3.5 and above (Kepler GK110b). However, on-chip memory cannot be passed to these functions, so they still need to operate on global memory pointers, i.e. to transfer data from and to the global memory. Starting with CUDA 6.0, it now also supports multi-GPU operations with a single launch from the host, handling automatically data division and transfers, irrespective of the matrix size, which only needs to be allocated on the host. This feature is called cuBLAS-XT API, is only implemented for level 3 BLAS functions (matrix-matrix operations). Currently this feature is only fully available with CUDA 7.0 and higher.

cuSPARSE contains the basic linear algebra subroutines for sparse matrices. It is used in the same manner as cuBLAS, but it does not provide device functions and automatic multi-GPU parallelization. It supports the basic sparse matrix and sparse block matrix formats.

#### 4.1.2.5 PCI express communications and NVIDIA GPUDirect

In devices with compute capability 2.0 or lower, data had to be transferred between GPUs and between GPUs and other devices, or other computer nodes by first copying them to the host memory. GPUDirect [117] is a technology introduced by NVIDIA in 2010 with the aim to enable faster communication between NVIDIA GPUs and third party PCIe devices, by avoiding copying data to the host before they can be transferred to another device. The first version was supported by CUDA 3.1 and RedHat Enterprise Linux (RHEL) for Fermi Tesla devices, and allowed communication with other PCIe drivers via shared pinned host memory. Supporting network devices vendors were Mellanox and Qlogic [118, 119], for use only with products that use the Infiniband protocol. GPUDirect version 2.0 was introduced with CUDA 4.0 and provided peer-to-peer memory access and Direct Memory Access (DMA) transfers between GPUs on the same PCIe bus. Based on the UVA memory model, GPUDirect Remore Direct Memory Access (RDMA) [117, 120] that was developed for CUDA 5.0 and Kepler GPUs, enables DMA communication between GPUs and other PCIe devices using standard features of the bus interface in the third party driver module. At present, the only hardware offering GPUDirect RDMA support is Mellanox Infiniband network adaptors, and in terms of software libraries, Message Passing Interface (MPI) libraries for distributed computing between nodes, now support the feature through the CUDA-aware MPI method [121]. There are also efforts for custom driver implementations for FPGAs [122, 123]. The general concept of GPUDirect is shown in figure 4.10. GPUDirect could be of interest to AO RTCS design because it eliminates the need for addi-

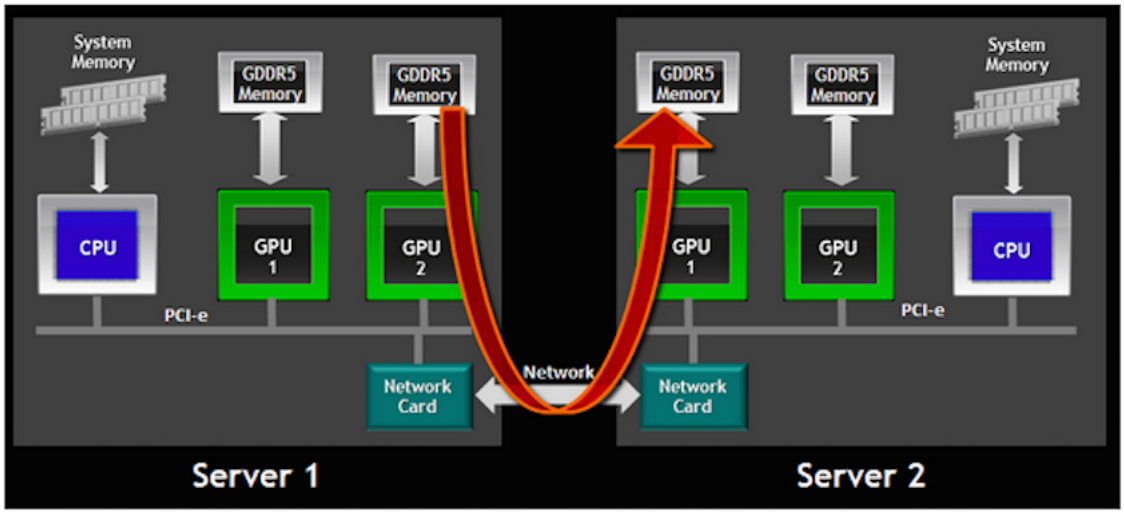



Figure 4.10: The GPUDirect concept. Data are transferred from the GPU to the network card directly using remote direct memory access (RDMA). (NVIDIA [117])

tional copies between the host memory and the AO devices, which is crucial for a low-latency system. Transferring data via the CPU memory adds both latency and jitter in the process. The additional latency depends on the size of the data being transferred, and while for tomography sensor measurement vectors it is negligible, for pixel data that are used to produce the measurements it could be more significant. Jitter is not well characterised yet for ELT scale systems, but the additional transfers have the potential to increase it. With the use of GPUDirect, if suitable device drivers are developed, we can enable the transfer of pixel data directly from the AO WFS detector to the GPU for processing all stages of reconstruction, without the involvement of the host memory.

### 4.1.3 OpenCL and third party GPU software

The increasing use of parallel computing with CPUs and accelerators in supercomputing has lead to the development of an open standard for parallel programming of

heterogenous platforms. The Open Computing Language (OpenCL) [124] specification was first released in 2009 by the Kronos Group, and was developed by Apple in collaboration with AMD, IBM, Qualcomm, Intel, and Nvidia. It specifies a standard for parallel computing across CPUs, GPUs, Digital Signal Processors (DSPs) and FPGAs among others, and it is the official language for AMD GPUs. The language is based on the C99 standard and provides a code framework that can be executed on different target architectures. OpenCL abstracts the architecture-specific elements of the code and has its own parallel programming model that maps to all types of targets. This however poses a restriction on architecture specific optimizations, so when porting CUDA programs, not all features of the CUDA model will be supported, and various CUDA compiler optimizations will not be available either, hence some performance sacrifice is to be expected, depending on the application code. CUDA no longer supports OpenCL in its software platform.

There are many third party software libraries that have been developed so far covering a large array of standard numerical operations, as well as software interfaces for different languages that can simply plug-in CUDA functionality. The wavefront reconstruction numerical operations often use LAPACK routines for operations such as matrix inversions, and two notable implementations of this library onto GPUs are the CULA library from EMPhotonics [125] and the Matrix Algebra for GPU and Multicore Architectures (MAGMA) open-source library from the Innovative Computing Laboratory at the University of Tennessee [126]. There also other projects implementing LAPACK routines on GPU, which can be found at the NVIDIA CUDA website, but these are the most complete so far. Extensions

for high level interpreted languages also exist, like the PyCUDA project [127] and the Numba open source compiler [128] included in the Anaconda scientific Python distribution for Python, and Matlab has its own Parallel computing toolbox.

## 4.2 AO Real-time Control Algorithms on GPUs

### 4.2.1 Latency requirements in AO

The requirements in execution time for the wavefront reconstruction operations, are set by the temporal behaviour of turbulence in the atmosphere, and the temporal response of the various components of the AO system. Greenwood [129] has provided a measure of the required correction bandwidth in relation to the rate of change of a Kolmogorov atmospheric turbulence, and the residual wavefront error achieved. The Greenwood frequency $f_G$ represents the characteristic frequency of atmospheric turbulence and depends on the turbulence conditions profile for a specific location. It determines the rate of response for an adaptive optics system in order to provide sufficient compensation. As a general rule, the correction bandwidth should be at least 10 times the Greenwood frequency [22]. Any delays in the response result in a temporal wavefront error, that is expressed in terms of $f_G$ and the bandwidth of the system $f_c$:

$$\sigma^2{}_T = \left(\frac{f_G}{f_c}\right)^{5/3} \tag{4.2.1}$$

Depending on the turbulence conditions, $f_G$ has a range between tens and thousands of Hertz, so in order for the AO system to maintain a good level of compensation and stability, the rate of correction should be between hundreds and thousands of Hertz.

The rate of response depends on the detector integration time, and the optimal AO response is a trade-off between this temporal error and the measurement error due to reducing the integration time [13].

An additional temporal error occurs due to the time delay that is required to process the data, which includes the wavefront reconstruction process. This error is also related to the Greenwood frequency[130]:

$$\sigma^2{}_{Td} = \left(\frac{t_s}{t_0}\right)^{5/3}, \quad t_0 = \frac{0.134}{f_G}, \tag{4.2.2}$$

where $t_s$ is the time delay due to data processing and $t_0$ the time constant which depends on $f_G$.

The effects of increasing latency on image performance have been studied [130, 131], and have been shown to have an increasing impact with spatial frequency, thus degrading the response of correction to high frequency wavefront errors. An example of the relationship between the residual wavefront phase variance, the atmospheric conditions, and the time delay in an AO system can be seen in figure 4.11. According to Roddier [14], the temporal variation of the residual wavefront variance is characterised by an atmospheric temporal cut-off frequency, with the residual variance increasing much more rapidly above it. The cut-off frequency is related to the Zernike polynomial radial degree $n$ and is given by:

$$f_{cut-off}(n) \approx 0.3(n+1)\frac{\bar{v}}{D_t}, \tag{4.2.3}$$

where $\bar{v}$ is the average wind velocity along the propagation path and $D_t$ is the

telescope diameter. The graph of figure 4.11 shows the effect for the tip/tilt mode, and illustrates that higher cut-off frequencies generally degrade performance because of the limited response time of the AO system, and that the additional time delays for computations also have a separate significant effect in reducing performance for a given cut-off frequency. It can be seen that the residual variance increases by approximately a factor of 6 when the latency increases from a quarter of a frame to one frame.

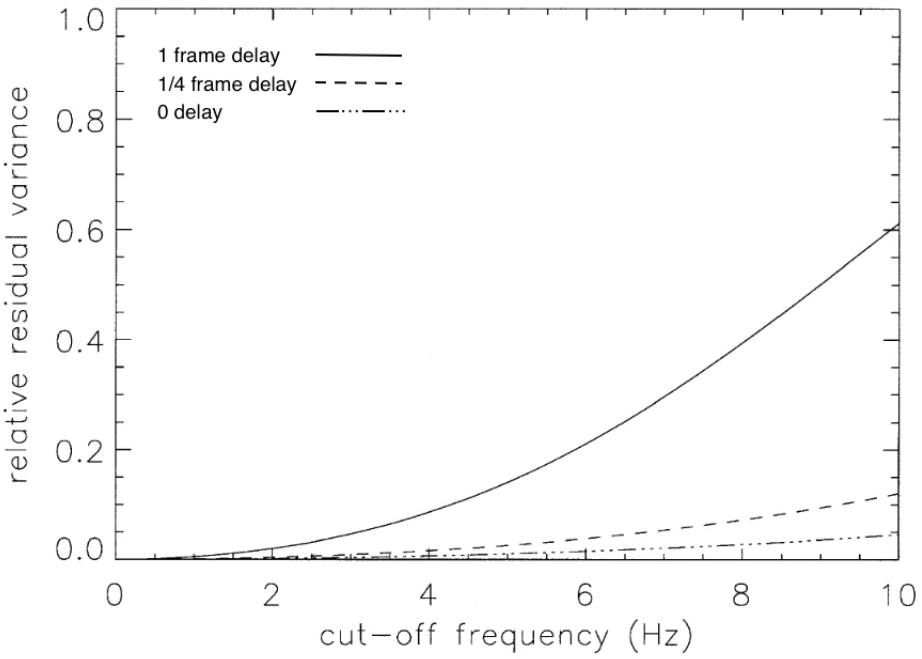



Figure 4.11: Relative residual phase variance versus atmospheric temporal cut-off frequency for various time delays (Roddier [14]). The cut-off frequency corresponds to the temporal variation of the tip-tilt mode.

Figure 4.12 shows a breakdown of timings for an AO system. The minimum time needed by the system to apply a correction from the moment the wavefront is received at the detector, is determined by the science requirements and limitations of the electronics in use. The time required for an image measurement to be available for processing, is the detector integration time, which is affected by the desirable

measurement accuracy and the detector response characteristics. Technical delays occur from the Digital to Analog Converter (DAC), the rise time of the High Voltage Amplifier (HVA) used to drive the mirror, and the mirror settling time. If we set the detector frame acquisition as a time constant, the minimum total time delay that is inherent by the science and system components is then equal to one frame. This implies that we exclude the delays associated with reconstruction computations and data communications, and assume a perfect amplifier and mirror. If these costs are considered, then an average minimum delay time of 2 frames can be assumed [13], allowing half frame for the reconstruction computations. For the control computer, we define as latency the time between when the last pixel is received by the computer system and when the last command arrives at the deformable mirror [132]. The

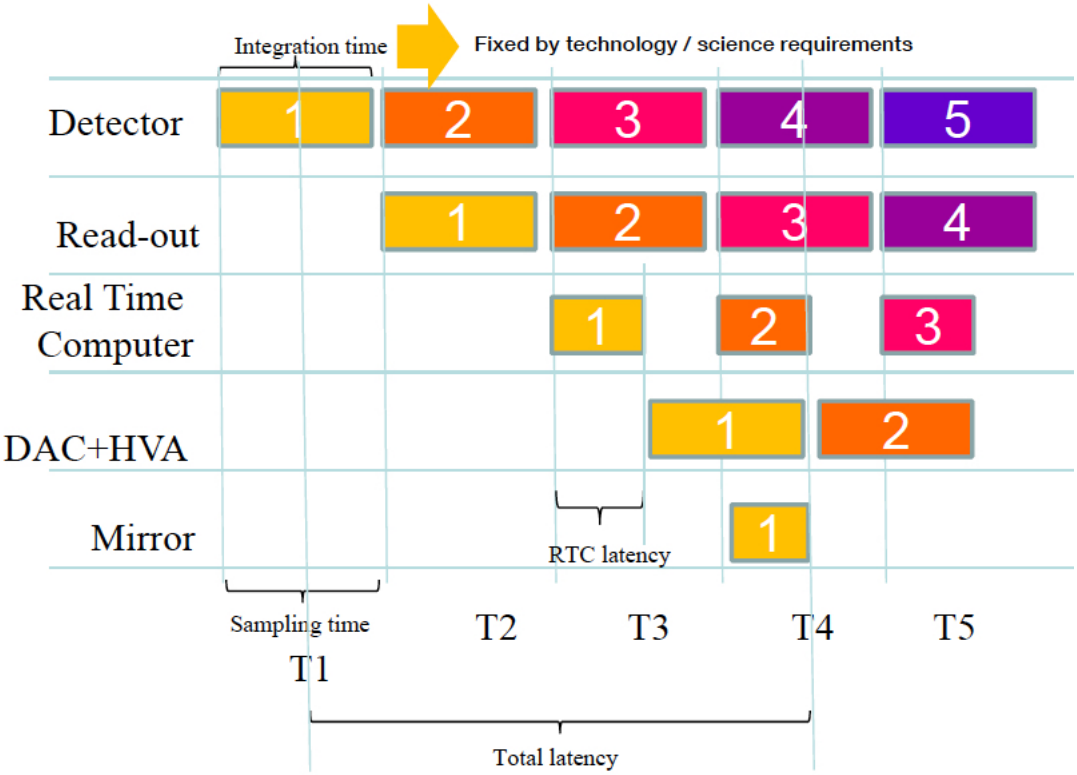



Figure 4.12: Adaptive optics operation timeline (L.Pettazzi [133]).

time that can be dedicated for computations depends on the system design and specifications. The maximum computation time to achieve the image performance specifications should generally be no more than half frame [133]. For a system such as MAORY, which is expected to have detector frame rates between 500 - 700 Hz [134], this corresponds to a latency of up to 1ms for all control computations. Atmospheric tomography reconstruction then is restricted to less than 1ms, in order to include pixel processing and gradient calculations.

### 4.2.2 A Real-Time AO Pipeline

The AO Real Time Control Pipeline (RTCP) performs the computational work required to reconstruct the atmosphere distorted image in real-time, by taking as an input WFS camera pixels, and delivering control vectors to be sent to DMs. An example of a working RTCS system is the Durham AO Real-time Controller (DARC) [135, 136], which is implemented in parallel using multiple threads on a CPU system, but also has the ability to use GPUs. Figure 4.13 compares the two processing strategies particular to DARC, which are important in order to minimise latency in a multi-threaded environment. The vertical strategy is the most conventional of the two, in which each thread is assigned a separate task. With the horizontal strategy, all threads perform all the tasks on different sub-apertures, as they arrive. The horizontal strategy has the advantage of improved load balancing between processors and avoids the need for synchronisation and communication between threads that exists in the vertical strategy, thus reducing latency. A disadvantage of the horizontal strategy is that it is less flexible in using different hardware for separate processing

stages. For a GPU to be efficient, a substantial amount of data must be delivered to it for processing, so a sub-aperture by sub-aperture processing would not perform well. There is however a good potential with this strategy to find an optimal amount of pixels to be processed by each thread in a tomographic system configuration, in order for the data to be sent to multiple GPUs, using one GPU per thread.

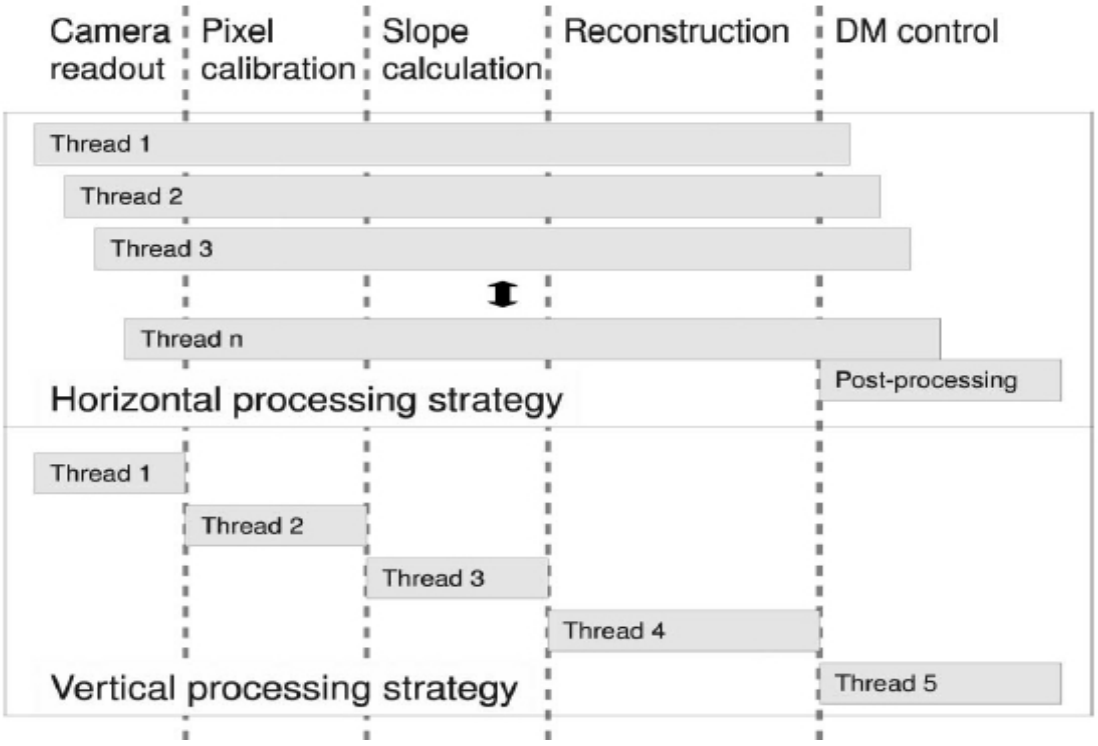



Figure 4.13: Adaptive optics real-time control pipeline for DARC (Basden et al. [135])

### 4.2.3 Requirements for Parallel Operations

In order to define computational requirements, the current specification of a known MCAO system under development for the E-ELT is used in this study. The Multi-conjugate Adaptive Optics RelaY (MAORY) MCAO module [137–139] specifications are used as a model for the computations in this study, and are listed in table 4.1. The CFAI was part of the MAORY module Consortium [140], working on the Real-

time control system, and Durham AO Simulation Package (DASP) simulations of MAORY have also been developed as part of this [139].

| HO WFSs | corr. layers | total subaps | acts DM1 | acts DM2 | acts DM3 |
|---|---|---|---|---|---|
| 9 | 3 | 90992 | 4408 | 1480 | 1813 |

Table 4.1: Specifications used for MCAO tomography computations, based on a MAORY DASP simulation.

The ideal goal would be a full end-to-end parallel or GPU parallel AO RTCS. This study is however concerned only with the wavefront reconstructor operations, that dominate the execution time in systems with large number of subaperures [141].

Memory storage becomes important as the degrees of freedom increase, and with tomography it also increases with the number of wavefront sensors and atmospheric layers the correction is applied to. Figure 4.14 shows an estimate of the increase in required memory storage for a full reconstruction matrix with the number of WFS, layers and WFS subapertures. In our case study, storage for the matrix in dense format is 2.6 GB, which fits on a GPU of Fermi generation or greater. The computational power required for a matrix-vector multiplication in terms of floating-point operations depends on the number of rows and columns and if the control matrix is sparse, on the number of non-zero elements per row. The dense reconstructor's dimensions from the above example is $90992 \times 7701$ and requires approximately an equal number of floating point operations. The MAORY study specifies so far a minimum WFS integration time of 2 ms for its high order control loop [139], hence the matrix vector operation would need at least 350 Gigaflop/s. Since the operation is bandwidth limited, each GPU card is restricted by its memory bandwidth. Experiments with GPUs have shown that one dual-GPU NVIDIA K80 card delivers about

45 Gigaflop/s for this particular operation. If we assume strong scaling and neglect possible inter-node communication overhead, it is strongly expected that this goal can be fulfilled using 8 of these GPUs. However, scalability can be limited by the computer system configuration, and systematic measurements must be performed to determine this, which hasn't been possible for this study because the resources were not available.

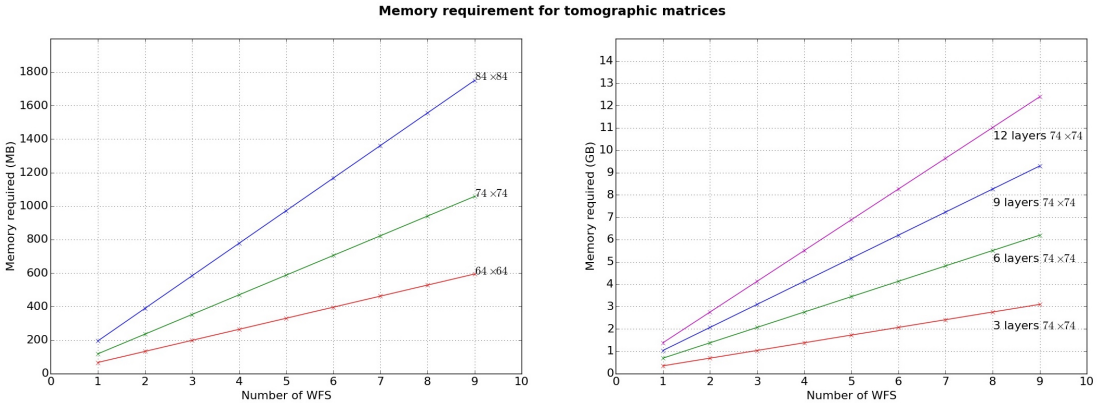



Figure 4.14: Memory storage requirement for dense tomographic matrices

Parallelizing the matrix-vector operation is straight forward, and is accomplished by dividing the matrix into sub matrices in the direction of the rows, and then combining the result in one vector. A synchronisation barrier is required to ensure all sub-matrix multiplications have finished before the result is transferred to the next stage. This is the case both when using multi-core CPUs with multi-threading, and when multiple GPUs are used.

Conjugate gradients iterations could be easily partly parallelised, by dividing individual operations as above. However, while this could be beneficial on a CPU platform, it could prove less efficient on a GPU, because of the memory transfers required before and after each algebraic operation, even when data are fully processed on the GPU memory, therefore the focus should be in minimising these transfers.

In addition, there will be a multiple synchronisation overhead, although that may be less significant.

# Chapter 5

# Mapping operations on a GPU

This chapter describes how matrix-vector multiplications and conjugate gradients can be performed on the GPU to exploit parallelism on these devices. An example configuration is analysed, and problem areas are identified and examined for each case.

## 5.1 Matrix-vector multiplication

Assuming the pipeline allows one additional frame delay initially, that will allow a full frame of data to be received and processed at once, the matrix-vector multiplications can be performed on-line in three steps. It is also assumed that the control matrix has been produced offline using SVD and has been pre-loaded on the GPU.

1. Transfer gradient vector to the GPU(s) via PCIexpress
2. Launch cuBLAS SGEMV (MVM) with pre-loaded matrix on GPU(s)
3. Transfer command vector back to the CPU(s) via PCIe

The PCIe transfers can be overlapped by running asynchronous computations and memory transfers on the GPU using streams. The overlapping allows for a virtually zero transfer overhead, as can be seen in 5.1. We can also see that the vector transfer time is negligible compared to a cuBLAS MVM on that vector, so even overlap is perhaps not necessary in this case, although it is probable it would be necessary if the GPU computations included the whole of the RTC operations, where pixel data would be uploaded. However, for AO, DMA transfers would be more desirable in order to connect directly input and output from other devices, avoiding intermediate data copies to the host system memory, and also to minimise other effects associated with using the system memory, such as operating system jitter [133, 135, 141]. To increase speed, we can apply the MVM using multiple

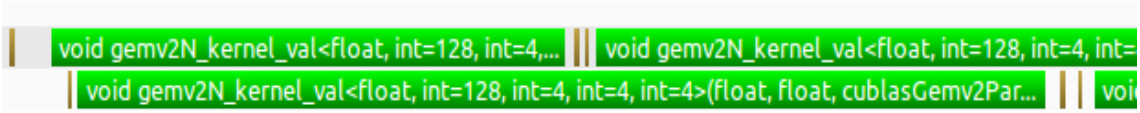



Figure 5.1: Overlapping GPU computations with memory transfers using CUDA streams. Snapshot from an nvvp profiler run of streamed memory copies and cuBLAS SGEMV launches.

GPUs, which was shown to provide a good scaling with the number of GPUs in earlier tests (figure 5.2).

## 5.2 Conjugate Gradients Iterations

The CG algorithm basic steps are recalled here, including the accompanied GPU tasks to be performed. A sparse matrix formulation is assumed, and the matrix is stored using Compressed Row Storage (CRS) format [142]. The conjugate gradients iterations can be easily implemented on a GPU using two CUDA libraries, cuS-

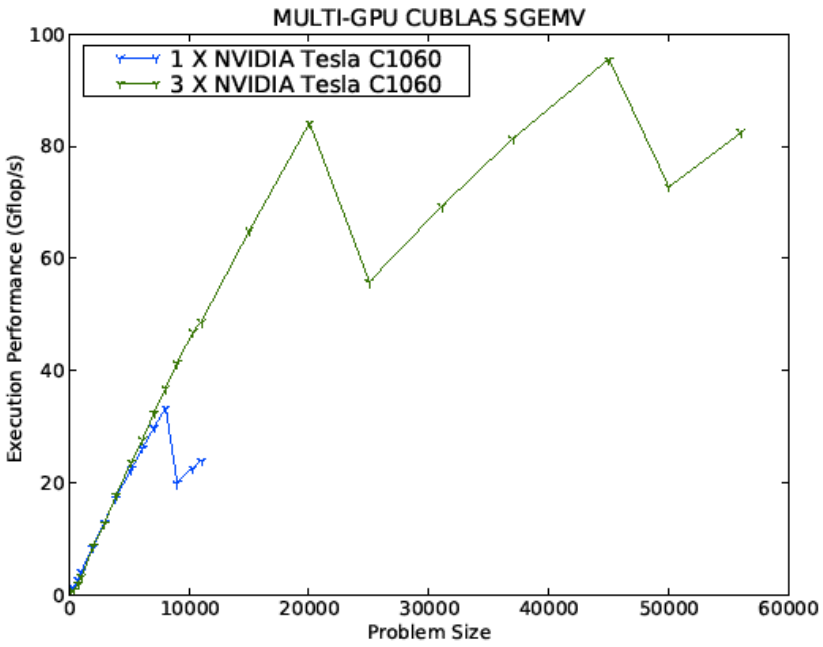



Figure 5.2: Performance in GigaFlop/s of cuBLAS SGEMV on 1 and 3 Tesla C1060 GPUs with increasing number of DM actuators. The sudden drops in performance result from a size dependency in execution time that existed in earlier versions of the cuBLAS library on older devices. This is not present in current hardware and library versions.

PARSE and cuBLAS. Each operation is therefore launched separately. The matrix is also pre-loaded on the GPU.

1. Transfer gradient vector $\mathbf{b}$ and previous frame solution $\mathbf{x_0}$ on GPU via PCIe
2. Initialisation : Perform $\mathbf{r_0} = \mathbf{b} - \mathbf{A}\mathbf{x_0}$, $\mathbf{p_0} = \mathbf{r_0}$, $r_1 = \mathbf{r_0}^\mathbf{T}\mathbf{r_0}$ using cuSPARSE and cuBLAS
3. Repeat:
   (a) $\mathbf{ap} = \mathbf{A}\mathbf{p_k}$ (cuSPARSE SGEMV)
   (b) dot $= \mathbf{p_k}^\mathbf{T}\mathbf{ap}$ (cuBLAS DOT )
   (c) $\alpha_k = r_1/dot$, $\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha_k \mathbf{p_k}$ (cuBLAS AXPY)
   (d) $\mathbf{r_{k+1}} = \mathbf{r_k} - \alpha_k \mathbf{ap}$ (cuBLAS AXPY)

(e) $r_0 = r_1, \quad r_1 = \mathbf{r_{k+1}^T r_{k+1}}$ (cuBLAS DOT)

(f) If $r_1 > r_{min}$, exit loop.

(g) $\beta = r_1/r_0, \quad \mathbf{p_{k+1}} = \mathbf{r_{k+1}} + \beta \mathbf{p_k}$ (cuBLAS AXPY)

4. Transfer result $\mathbf{x_{k+1}}$ to host via PCIe

Although the two libraries are very efficient, the speed of the computations on the GPU is limited by the transfers to and from the device global memory. The read/write instructions on a GPU currently have a latency of 200 - 400 clock cycles. This means that when operations are dependent on data residing in global memory, then execution is stalled per warp until the operands are available. If maximum data parallelism is applied, and the rate of computations to global loads and stores is low, then the execution speed is limited by the global memory bandwidth. As can be seen in figure 5.3, the CG algorithm implemented using CUDA libraries as described above, requires loading of **A** once, and loading and storing vectors the size of the command vector 10 times and 4 times per iteration respectively. If the CG is to reach convergence in $n$ iterations, the command vector length is $N$ and the number of non-zero elements in **A** is $nnz$, then the total number of element global memory transfers required per AO iteration is:

$$M \approx nnz + 4 \times N + n \times (nnz + 14 \times N) \tag{5.2.1}$$

It is therefore desirable to reduce this number to the number of initial vectors and matrices needed, and them reuse them recursively and only transfer the final result.

Depending on the structure of **A**, and the number of non-zero elements per row,

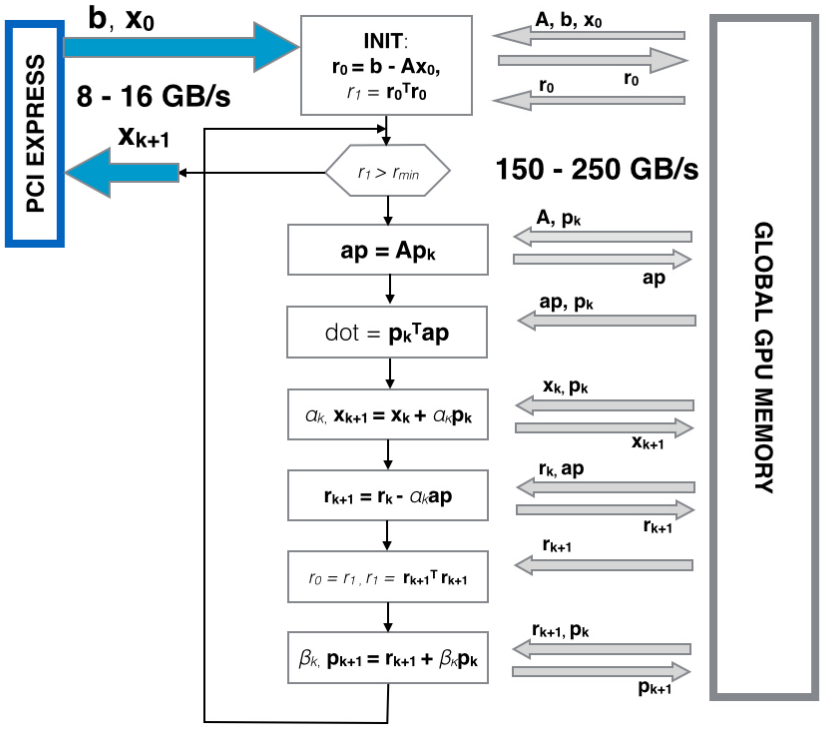



Figure 5.3: Illustration of data movement during the CG process on GPU, using library routines for linear algebra operations. The size of each vector is the the size of the combined tomographic layers vector.

there is a possibility for the operations to be combined in a single GPU kernel. In the most general case where the sparse matrix is stored in a CRS format, it can be accessed in a custom manner, and the computations can be done manually. cuBLAS device library functions are not recommended in this case, as they still need to separately load data from global memory. The GPU kernel then has the following structure:

1. Initialisation: As before, using cuSPARSE and cuBLAS, for simplicity. It happens only once so it can be considered a constant small overhead.

2. Begin custom kernel:

   (a) Load partial **A** per GPU thread block from global memory. Each thread operates on one row of **A**, data loaded on registers per thread, or on

shared memory. $\mathbf{A}$ is stored in a CRS format, so only the non-zero elements and their indices are loaded. Their number depends on the sparsity of the matrix.

(b) Load $\mathbf{p_k}$ for the matrix-vector product, each thread loads elements with indices for processing the respective row of $\mathbf{A}$. Partial $\mathbf{p_k}, \mathbf{r_k}$ are also loaded on shared memory, or as element per thread, to match the number of rows processed in a single block. This only happens once.

(c) Repeat:

i. $\mathbf{ap} = \mathbf{Ap_k}$: Each thread loops over a row of $\mathbf{A}$ and the equivalent stored elements of $\mathbf{p_k}$, result $\mathbf{ap}$ is held on shared memory.

ii. Perform partial dot product of dot = $\mathbf{p_k^T ap}$ and combine partial sums using a synchronization mechanism within the kernel. That implies synchronisation between all blocks when partial sums are finished.

iii. Calculate $\alpha_k$ and partial $\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha_k \mathbf{p_k}$ : element per thread, result is written to existing on-chip storage, requires a floating-point division per thread.

iv. Perform partial $\mathbf{r_{k+1}} = \mathbf{r_k} - \alpha_k \mathbf{ap}$, as in 2(c)iii

v. Do $r_0 = r_1, \quad r_1 = \mathbf{r_{k+1}^T r_{k+1}}$ using the process in 2(c)ii.

vi. If $r_1 > r_{min}$, exit loop.

vii. Calculate $\beta_k$ and partial $\mathbf{p_{k+1}} = \mathbf{r_{k+1}} + \beta \mathbf{p_k}$ as in 2(c)iii.

End custom kernel.

3. Finalization: Transfer result $\mathbf{x_{k+1}}$ to host via PCIe

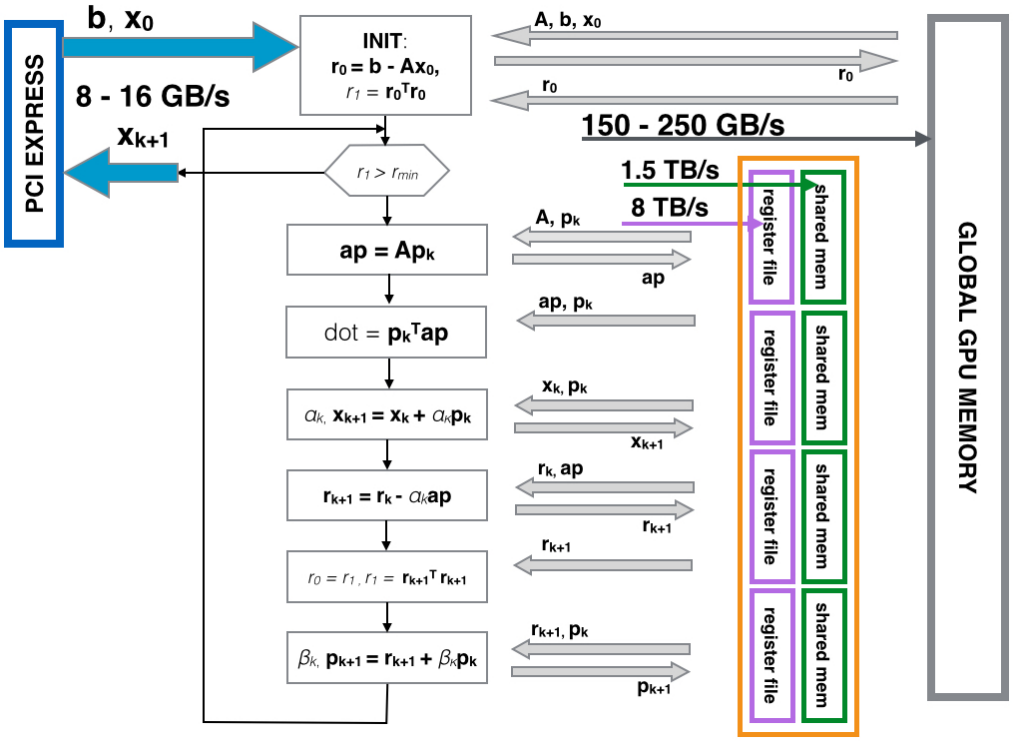



Figure 5.4: Illustration of data movement during the CG process on GPU, using custom computations with on-chip memory. Note the difference in bandwidth between different memory spaces

An illustration of the above process is shown in 5.4, where the difference with 5.3 can be seen. Of course the size of the on-chip memory spaces is very small (64 K registers, 48KB shared memory per SM) compared to the global memory (6 - 12 GB), and eventually data has to be loaded in chunks that fit the on-chip resources and be reused as much as possible before it's necessary to write them back to global memory. In analogy with (5.2.1) the potential minimum amount of full global memory element transfers can now be reduced to:

$$M \approx nnz + 4 \times N \tag{5.2.2}$$

The initialisation step can alternatively be included in the kernel, using the same

storage, and the same process for the operations as in the main body of the kernel. This could potentially save extra time from memory transfers. Using this type of scheme becomes expensive in terms of local storage and synchronisation barriers, but avoids multiple global memory transfers per iteration. There is a trade-off between multiprocessor occupancy and use of on-chip resources. If there is a high usage of shared memory or registers per thread block, then the number of blocks or warps executed in a single multiprocessor is limited to fit the SMX local resources, therefore, thread parallelism becomes limited. Depending on the global memory instructions and the arithmetic instructions arrangement in the code however, the performance achieved differs, and it is best verified by experimenting and profiling the application. The requirements in local on-chip storage per thread block depends on the block size. To maximise parallel execution, the number of threads per block should be a multiple of 32 (one warp). The full vectors, ideally need to also fit the absolute amount of available resources so that the process can be parallelized on the full size of the problem, and not requiring a block matrix algorithm. Assuming a block size of 32 for this kernel, a tomography vector size of $N \approx 50000$ (9 layers, $74 \times 74$ sub-apertures), and a Kepler device with 15 SMX, such as the K40, requires as a minimum:

- Registers:

$$32 \times (2 \times nnz/\text{row}), \qquad \text{for } \mathbf{A}, \mathbf{p_k}$$

- Shared memory per block:

$$32 \times 3, \qquad \text{for } \mathbf{p_k}, \mathbf{r_k}, \mathbf{ap} \text{ in their full form}$$

- Shared memory per SMX:

$$3 \times N/\text{SMX} \approx 40\text{KB}, \qquad \text{for } \mathbf{p_k}, \mathbf{r_k}, \mathbf{ap}$$

Both registers and shared memory storage are very demanding. The registers per thread, depend on how sparse the matrix is. Each thread can have up to 255 floating-point registers, but as the amount of registers used increase, so does the pressure on the multiprocessor and the number of warps that can be allowed to execute. The compiler may decide to relieve the pressure by off-loading data to a per-thread space in global memory, in which case the execution will suffer from global memory transfer delays. If the data are loaded to shared memory instead, then the amount of data per block needed in order to be accessed by threads in parallel are at the order of several kilobytes per block, depending on how sparse the matrix is. The maximum amount of shared memory per block and per multiprocessor is 48 KB, restricting the number of blocks that can be executed using this memory space. By storing only the vectors on shared memory, the data size is so big that it would only allow one block per SM, but it is doable, and may still be faster than the library version. In addition, future generations of GPUs will have increased register file and shared memory size, as mentioned in 4.1.2.3, that would allow higher parallelism. For the MAORY simulation case studied here, we only correct 3 atmospheric layers, so the vector sizes are 3 times smaller and the pressure on the device on-chip resources is much lower. The MAORY configuration will instead need up to 10KB per SM, and, in practice, as the layers correspond to conjugated DMs in this case, the pupil sizes for each DM height decrease the number of actuators used. In the DASP MAORY

simulation used in this study, the combined DM vector size is 7701, which accounts for about 6 KB on-chip memory usage. Some important limitations are the number of non-zero elements per row, which for the given configuration can become too large, and the need for indirect memory accesses in the CRS format, which does not fit the GPU hardware model. Block matrix techniques would be beneficial, where blocks can be arranged so as to be computed independently, keeping the principal of holding the data local to the chip and reusing them in the CG iterations. Figure 5.5 shows the parallel operations on the GPU schematically.

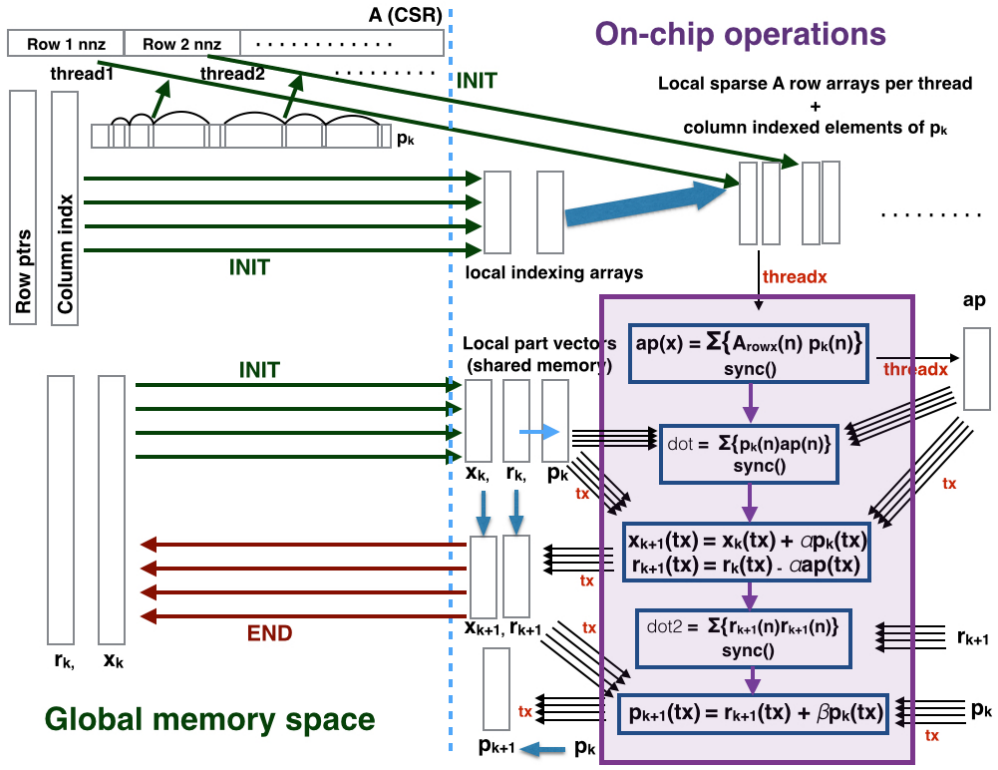



Figure 5.5: Parallel CG execution on the GPU: each thread loads the non-zero elements of a row and their indices on its registers (or shared memory), and the required elements for each row from $\mathbf{p_k}$. It then performs CG iterations operating on rows and result vectors in parallel. Blue arrows denote in-place computations (k + 1 replaces k in memory), a single black arrow is a single thread, and multiple black arrows represent parallel thread operation on elements. Global memory reads are indicated by the thick green arrows, and final result global memory write by the thick red arrows.

The same scheme can be used with a preconditioned conjugate gradient application, with the preconditioning step introducing a specific to the preconditioner type of operation.

# Chapter 6

# Experiments on GPUs

In this chapter, I present the experiments done to assess how GPUs perform on an Adaptive Optics Real Time Control System scenario, when using a direct and an iterative solver. I am assessing effects on image correction quality, and examining trade-offs with computational speed on simulation for the iterative approach. I also report on execution speed for different GPU generations, and analyse the impact of GPU usage on system jitter with a set of benchmarks, followed by a discussion on the results.

## 6.1 Overview of Experiments

Two types of tests were conducted. The first used the Durham AO Simulation Package [143, 144] to simulate a full open-loop tomographic MCAO system, in order to verify the resulting image performance using GPUs by applying the control solution both with matrix inversion and Matrix Vector Multiplication, and with Conjugate Gradients iterations. In the case of CG, the maximum sparsity and minimum num-

ber of iterations required to achieve the acceptable wavefront error level was assessed. The second test consisted of benchmarks that measured the timing performance of MVMs and CG on different GPU configurations. Finally, jitter measurements were made for each case separately.

## 6.1.1 Experimental Setup

### 6.1.1.1 Hardware setup

The tests were conducted on two machines. The first is a dual socket 10 core Intel Xeon CPU E5-2660 at 2.60 GHz with a total of 40 threads and three NVIDIA Kepler generation GPU cards: 2× NVIDIA Tesla K20 and 1× NVIDIA Tesla K80 which contains two GPUs. The second is a dual socket 6 core Intel Xeon CPU E5645 at 2.40GHz with a total of 24 threads and 3 NVIDIA Fermi generation GPU cards: 3× NVIDIA Tesla C2070. For the CPU measurements only the Intel Xeon E5-2660 (Haswell) was used, as it is the latest architecture and the most powerful of the two CPU processors available.

### 6.1.1.2 Simulation setup

A schematic of the AO system in test is shown in figure 6.1. It models an AO system with incoming light through 9 atmospheric layers from 6 Laser Guide Star (LGS), 3 high-order Natural Guide Star (NGS), and 2 NGS that are used to measure low-order perturbations (overall tilt). This was made using the DASP simsetup utility, which allows the user to design a simulation graphically. Each of the square boxes represent a functional module, for example the box labelled “wfscent” is the

module that performs wavefront sensing and produces centroids which are sent to the "tomorecon" module that performs the tomographic reconstruction. A box frame around modules allows one to group many objects of that module. In this case we have a 9 atmospheric layers phasescreen ("infScrn") whose output is processed by an atmospheric pupil module ("atmos") that creates the projected phase from the different guide star directions through the different atmospheric layers to the telescope pupil. The wavefront sensor box ("wfscent") contains the 9 high order wavefront sensor objects, one for each guide star direction on the sky. Next the sensor data are passed to the tomographic reconstructor that combines the measurements and also uses the combined open-loop interaction matrix and atmospheric and noise covariances to produce the solution. There are several solver options in this module, and this test used a Minimum Variance Reconstructor produced with an LU (Lower Upper triangular) decomposition inverse [145] and solved with dense MVM, and a sparse formulation of the same, solved with CG.

#### 6.1.1.3 Benchmark setup

This test uses C CUDA code that reads input from the system matrices tested with the simulation, as well as vectors of measurements produced by the reconstructor, to perform and time individual linear algebra operations on the GPU. The cases tested are the matrix inversion and dense and sparse matrix-vector multiplications with the NVIDIA cuBLAS and cuSPARSE library, performed with one and multiple GPUs of the Fermi and Kepler generation, and are compared to CPU measurements with the Intel Math Kernel Library. Performance of the CG algorithm is then tested using the best performing sparse MVM case.

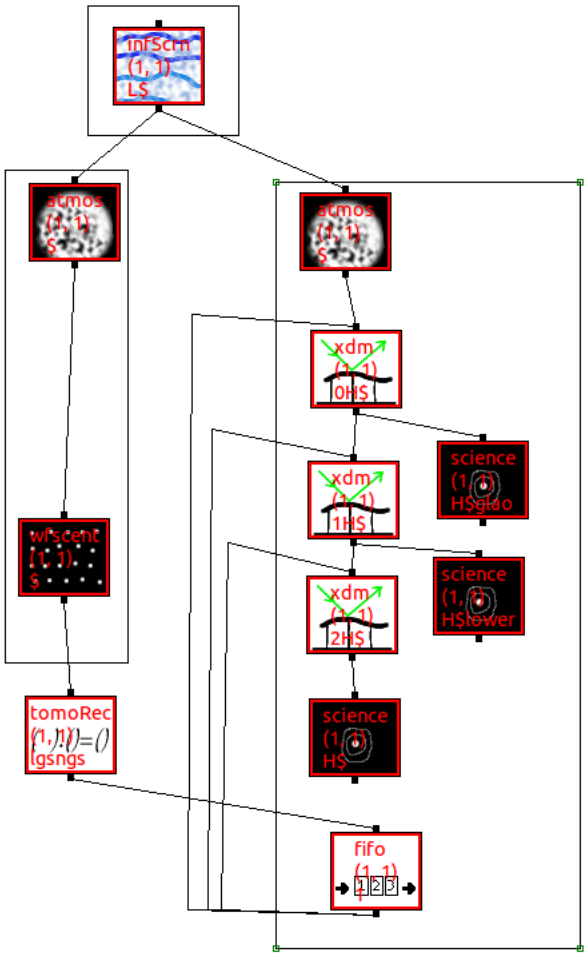



Figure 6.1: A schematic diagram of the AO simulation made using the simsetup utility of the DASP package (credit: Alastair Basden).

## 6.2 Experimental Results

### 6.2.1 Simulation results

We performed 5000 AO iterations to obtain convergence of results. The results from the LU reconstructor are used as a reference for imaging performance. Average execution time measurements of MVM and CG reconstructor operations were also obtained, to provide a measure for the formulation of a trade-off between imaging and timing performance. It should be noted that the software is not currently ca-

pable of simulating time delay effects on image quality, thus we are only looking at the effects that occur from modifying the coefficient matrices structure and sparsity on the static result. Phase and noise covariance statistics were used to form the reconstruction matrices. The sensor noise was considered as independent, uncorrelated measurements and was represented by a diagonal matrix of equal diagonal values, while the phase covariance operator was obtained as a sparse scaled Laplacian squared matrix approximation to the inverse phase covariance matrix, using a 5-point $3 \times 3$ stencil, following [26] .

#### 6.2.1.1 Initial Results

We perform dense matrix-vector operations on the system using a pre-computed baseline reconstructor calculated with the LU method. Prior statistics are incorporated in the construction of the inverse matrix for which LU factorization was used. We also use the sparse formulation to perform CG. It should be noted that in the case of the inverse matrix it has not been beneficial to use a sparse MVM, as the number of non-zero elements was quite high, and both performance and memory storage was worse than using a dense MVM. The figure for the GPU MVM execution time corresponds to the best achieved, which used 2× NVIDIA K20 GPUs. The figure quoted for CG corresponds to the fastest GPU performance of a CG matrix at the initial level of sparsity, using 30 iterations. The Strehl and ensquared energy corresponds to the best achieved by the particular simulation setup, which only reconstructs 3 layers instead of doing full tomography and at 0.764 arcsec seeing Table 6.1 summarizes the results.

It is quite notable that the Intel MKL library on the Intel Xeon E5-2660 CPU

| Solver | % SR | | % EE (75 mas) | | Real-time Execution (ms) | |
|---|---|---|---|---|---|---|
| | H band | K band | H band | K band | CPU | GPU |
| **LU (MVM)** | 16.79 | 35.95 | 29.26 | 43.39 | 51.566 | 11.82 |
| **CG (30 iter.)** | 16.88 | 36.07 | 29.27 | 43.40 | 49.57 | 19.42 |

Table 6.1: Adaptive Optics performance using dense Least Squares Estimation with Minimum Variance Estimation using LU Decomposition and Conjugate Gradients.

performs only about 2.5 times slower than the GPU, which is much faster than in older CPUs and MKL versions, as well as other libraries, like for example the scipy numerical package for Python, which performs the same operation in 281 ms. MKL operations are multithreaded and perform very well, however, the execution time has been observed to vary up to 20%.

#### 6.2.1.2 Determining the optimal CG configuration

We measure the effect of increasing the sparsity on imaging performance, and derive the minimum number of CG iterations required for convergence of the AO system to an acceptable result. Figure 6.2 shows the level of Strehl ratio against the cutoff value of poke matrix elements. Figures 6.3a and 6.3b show the average execution time as a percentage of the initial value, and the linearity of the operation with the number of non-zero elements. The reference execution was on a NVIDIA Tesla K20 GPU. We can conclude from the above that a reasonable choice is to use a matrix with non-zero values down to $10^{-4}$, which saves about 15% in execution time, and does not appear to have any loss, in fact it is slightly higher, possibly because values very close to zero which have been filtered out from the poke matrix in the CG formulation could have been a result of accumulated rounding errors.

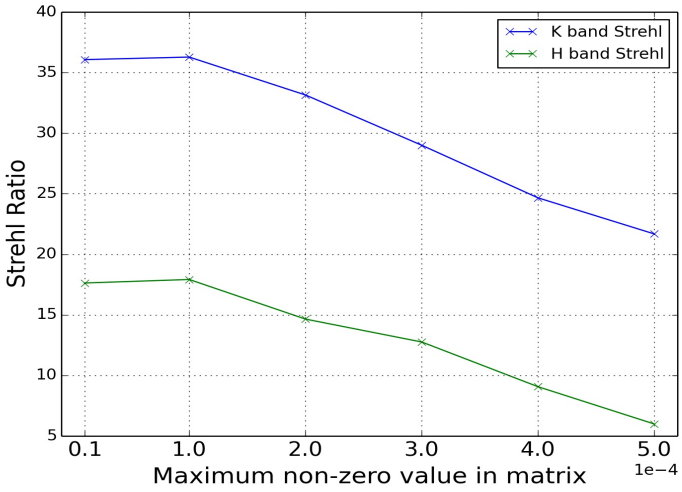
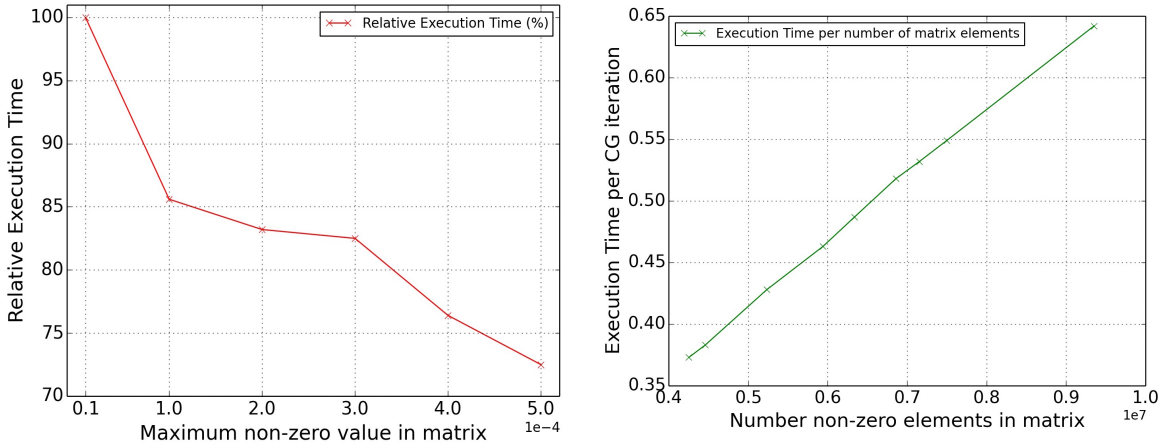



Figure 6.2: Conjugate Gradients solver with increasing matrix sparsity. Strehl ratio response to varying sparsity of the poke matrix .




(a) (b)

Figure 6.3: Conjugate Gradients solver with increasing matrix sparsity. Left: Execution time reduction for varying sparsity of the poke matrix. Right: Number of non-zeros in CG matrix and execution time per CG iteration appear to have an approximately linear relationship.

We also examine the minimum number of conjugate gradient iterations to run with this minimum sparsity choice, so that the resulting imaging performance is acceptable. Figure 6.4 illustrates the variation with decreasing number of iterations. This variation indicates that in the case examined we can reduce the number of CG iterations by 1/3 and by half with a loss of 1.23% and 3.93% in Strehl respectively in the H band, and 0.89% to 2.84% in K band. The Strehl ratio loss in percentage appears to be less in K band, while the values in absolute terms are more consistent

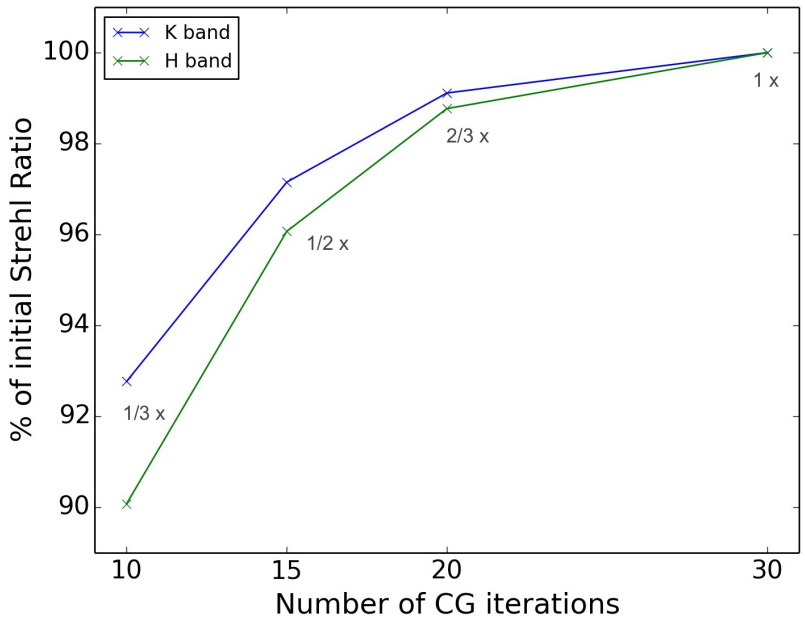



Figure 6.4: Effect of decreasing CG iterations on Strehl ratio. Labels at points indicate the fraction of execution time required.

between the two, and they follow a quadratic curve, which is characteristic of the convergence of the CG algorithm. The ensquared energy in all the above cases only falls up to a maximum of 0.48%. Table 6.2 shows the details in Strehl ratio.

Table 6.2: Strehl ratio in actual and percentage values for different numbers of CG iterations

| CG iterations | SR (H band) | % initial SR | SR (K band) | % initial SR |
|---|---|---|---|---|
| 30 | 17.04 | 100.0 | 36.28 | 100 |
| 20 | 16.83 | 98.77 | 35.95 | 99.11 |
| 15 | 16.37 | 96.07 | 35.24 | 97.15 |
| 10 | 15.35 | 90.08 | 33.65 | 92.77 |

## 6.2.2 Benchmark Results

### 6.2.2.1 Singular Value Decomposition

Although the simulation used a different, less computationally and memory demanding method to produce the inverse reconstruction matrix, we tested the SVD here for illustration purposes, to show how the classical solution performs computationally. Matrix inversion was performed using the SVD routine of the MKL library for the CPU and the CULA and MAGMA libraries on the GPU. This test used only one GPU, as currently these libraries do not support readily a multiple GPU formulation, and the off-line SVD operation is of lower interest to this study, therefore a custom multi-device formulation has not been attempted. Consequently, these results (Table 6.3) are only indicative of the speedup we can obtain for the off-line SVD operation using only one GPU.

Table 6.3: Execution time of SVD routines in CULA, MAGMA, and Intel MKL.

| CULA SVD | MKL SVD | MAGMA SDD | MKL SDD |
|---|---|---|---|
| 59.5 s | 97.9 s | 23.3 s | 49.1 s |

We can see again that MKL is only les than twice slower with the CULA library. In the case of SDD, a flavour of the SVD routine which uses a divide-and-conquer algorithm, we use the MAGMA library for comparison, which has the routine available while CULA does not have an SDD implementation. In this case we see again that the GPU is only less than twice faster.

#### 6.2.2.2 Sparse Matrix-Vector Multiplication

It has been observed that during a CG iteration, approximately 95% of the time is taken by the sparse matrix vector multiplication. Therefore it is meaningful to attempt the maximum speedup of this individual operation. However, it has also been found that for this particular case, there is a minimum time required from the rest of the operations that amounts to ∼1.7 ms, which is above the RTC goals as these were described for MAORY in section 4.2, and additional methods to parallelise these on a GPU should be sought.

The sparse matrix-vector multiplication is performed using 1, 2 and 3 Fermi GPUs, 1 and 2 Kepler Tesla K20 GPUs, and 1 and 2 GPUs on the Tesla K80 card. We do not use different cards to parallelize the problem, as they have different features which may introduce additional delays, and it is generally recommended that multiple GPUs of the same model are used for a single problem. Using the initial sparse CG matrix from the simulation, i.e a $7701 \times 7701$ sparse matrix with 9352293 non-zero elements (about 1 sixth of the full dense matrix), we break the operation into a number of parts equal to the number of GPUs we want to use. For the CRS format, some adjustments need to be made to apply the correct matrix section and row pointers. This is done at an initialization stage that uses a custom kernel which adjusts the value of the row pointers according to the section of the matrix being uploaded, and its computational cost is negligible. Execution time including PCIe memory transfers are reported in table 6.4. These values can be confidently extrapolated to the chosen sparsified matrix of the previous sections by subtracting 15% of the execution time.

Table 6.4: Execution time of sparse matrix-vector multiplications with cuSPARSE and Intel MKL.

| Device | Execution Time ($\mu s$) |
|---|---|
| Intel Xeon E5-2660 CPU | 1629.97 |
| 1× Tesla C2070 | 942.82 |
| 2× Tesla C2070 | 487.70 |
| 3× Tesla C2070 | 397.32 |
| 1× Tesla K20 | 524.14 |
| 2× Tesla K20 | 368.56 |
| 1× Tesla K80 | 524.75 |
| 2× Tesla K80 | 353.23 |

#### 6.2.2.3 Conjugate Gradients

The best performing single-GPU sparse MVM is incorporated into the CG routine. The reason for not using multiple GPUs for the CG iterations is that, in the CG algorithm, every step in a single iteration requires the complete result from the previous step (see 5.2). This must be available to the same GPU that is performing that step, and spreading computations across devices for each and every step may introduce further delays. These delays would result from the need to synchronise and download / upload intermediate data between CG iteration steps. This is also the case between CG iterations. Since, except the MVM, the rest of the steps have a very low computational cost, it is more efficient to perform the rest of the operations on a single device. The issue remains however, between the sparse MVM and the rest of the CG routine, because the total result must be made available to the GPU that will apply the rest of the computations. One way of achieving this is to perform peer-to-peer copies of the sub-vectors to the device that will take over. This incurs the cost of the synchronous data transfer through the PCI express bus. It was

estimated that overall such an implementation would only make little difference, and potentially increase jitter, so it was not attempted. The total best achievable performance of the CG using single-GPU sparse MVM including memory transfers for 15 CG iterations is approximately the same for the K20 and K80 devices, and is $\sim$ 10.5 ms, or $\sim$ 8.9 ms when using the sparsified matrix..

### 6.2.3 Jitter measurements

Computational jitter is a measure of the variability of the computational latency in multiple successive AO control cycles. It is a random variation added to the average latency, which results in instantaneous latencies that can be higher than the desired integration time. It can affect AO performance by causing multiple frames to have increased delay. The effect increases with the spread of the jitter distribution as a percentage of the nominal latency defined in the instrument performance requirements. For small spreads, the effect can be fully negligible. This, however, can vary significantly depending on the hardware, algorithmic and communication features of the RTCS, and there is a need for accurate characterisation. A useful study on jitter related to ELTs can be found in [133]. In this study, E-ELT simulations show (figure 6.5) that a jitter up to 1% is fully negligible, while there is some small observable Strehl ratio loss with 10% jitter. Jitter Above 10% appears to cause significant loss to the Strehl ratio, and the loss grows linearly with the amount of jitter, with a gradient that depends on the operational frame delay. Although there is not much experimental work to specify the acceptable jitter limit, and it is instrument specific, the experiments in [133] use 3 different frame delay cases, all of which seem to be

affected very little with up to 10% jitter, and this can provide a good indication that 10% can be seen as a critical upper limit value, and that instrumental designs should pursue jitter well below this value.

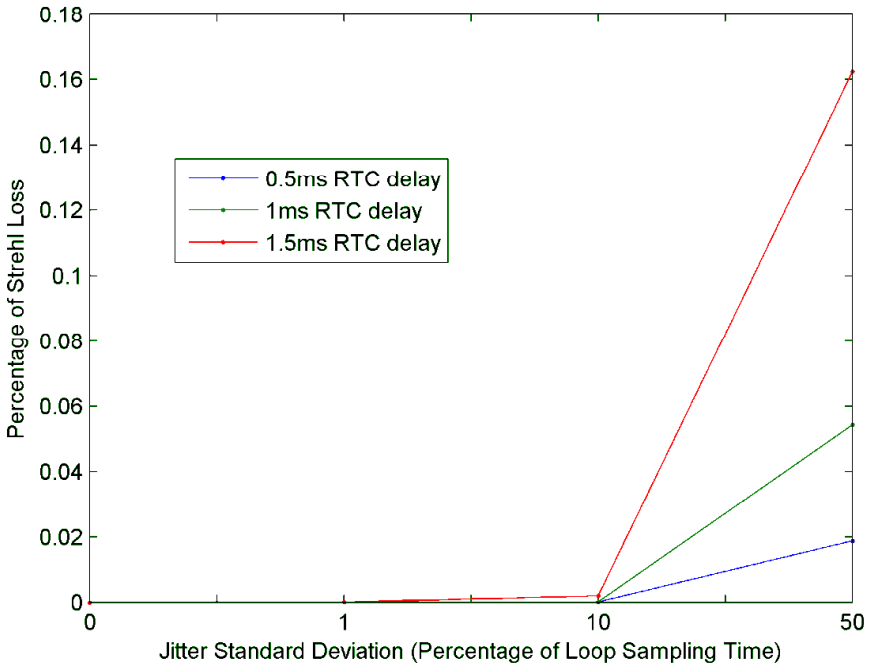


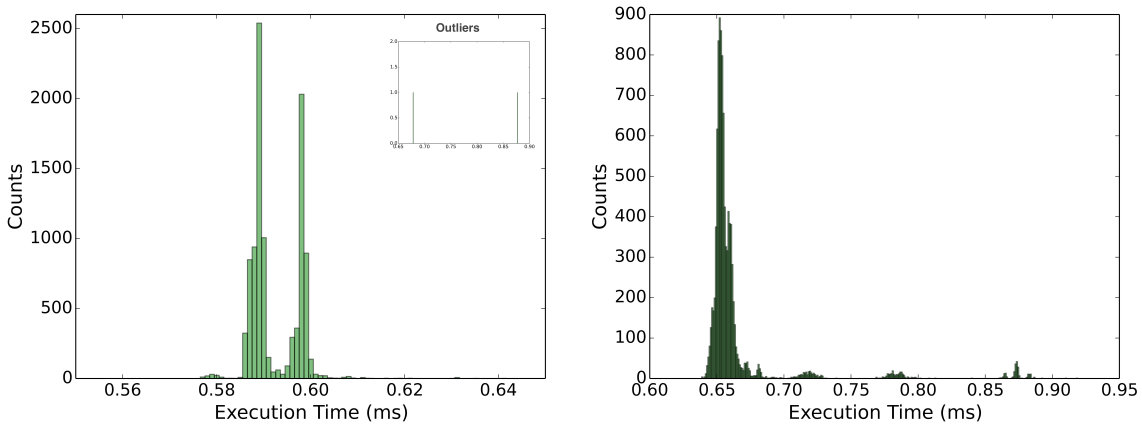

Figure 6.5: Results of Jitter Simulations by L.Pettazzi [133]

It is interesting to measure the jitter in execution latency when dealing with different hardware. Jitter measurements for GPUs have been previously performed for the Durham AO Real-time Controller [141], and it was found that for an NVIDIA GPU the jitter probability spread is much less than 1% which is smaller than what is commonly found on a CPU system. As multiple GPUs and asynchronous transfers and operations are employed in this experiment, we expand the jitter measurements to each of these operations. In the following sections we examine the jitter occurring in dense and sparse matrix-vector multiplications and PCIe transfers using small matrices that execute in the under 1ms regime, which is the requirement in AO. Further than this we also measure the jitter in executing 15 CG iterations. All

measurements were done on the dual 10 core Intel Xeon CPU E5-2660 (Haswell) and the three NVIDIA Kepler generation GPU cards ($2 \times$ K20 and $1 \times$ K80 (dual GPU)).

#### 6.2.3.1 Dense matrix vector multiplications

To limit execution time to the less than one millisecond region we used a $2500 \times 7701$ matrix (a 'slice' of the reconstruction matrix used to obtain imaging results). 10000 iterations of the process of vector transfer to GPU, cuBLAS matrix-vector multiplication (SGEMV), and vector transfer to the CPU memory, were run. Figure 6.6 shows the distribution of execution times from measurements on a single K20 and a single K80, using synchronous data transfers from and to CPU paged memory allocations.

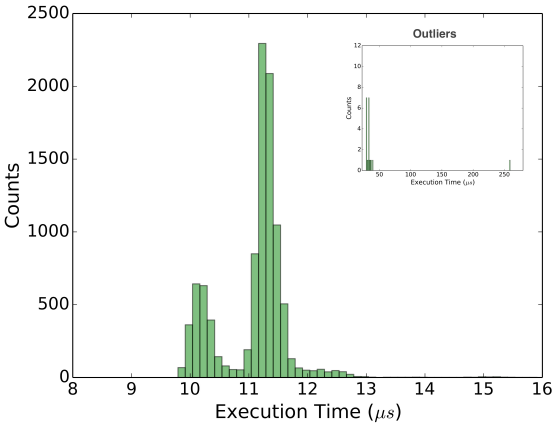



(a) K20 (b) K80

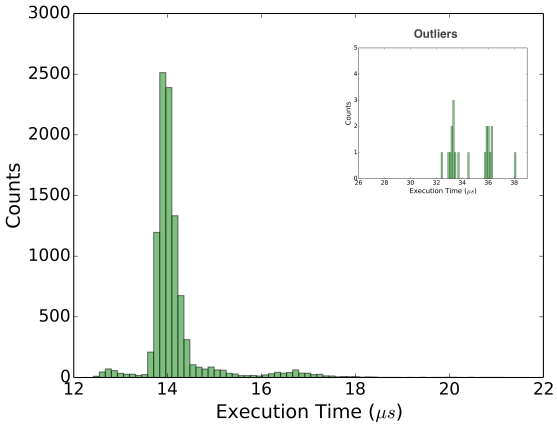
Figure 6.6: Jitter histograms for dense MVM on (a) a K20 and (b) a single K80 GPU.

On the K20, the histogram shows very few outliers, but the distribution has 3 peaks. This indicates that the distribution of the data transfers may be affecting the result. On K80 we observe an unusual pattern of variation. To examine these we measure the contribution of data transfers and SGEMV separately, to deter-

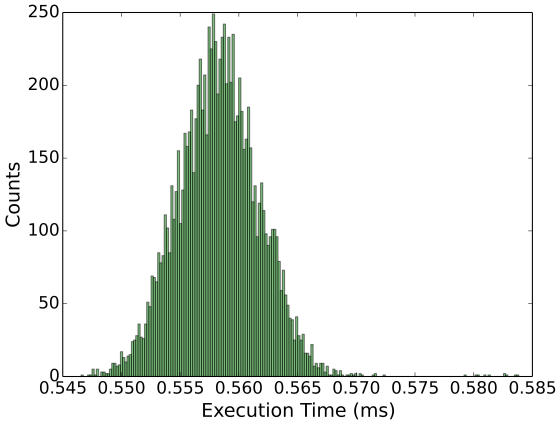
mine the main cause of jitter. In figure 6.7 we see the histograms of the separate measurements for the K20, which show that SGEMV is normally distributed with no significant outliers, while the data transfers carry a wider spread and occasional high outliers. The pattern of these occurrences can be seen on a line plot of the separate and combined measurements in figure 6.8. The K20 GPU supports only PCIe 2.0 generation, while K80 supports and uses a PCIe 3.0 link, and measurements with both cards were taken to locate possible effects of the different PCIe links. There we can clearly see that, for the K20, SGEMV has the smallest variation, while the data transfers have patterns that exhibit periodic spikes in duration. This behaviour can be attributed to the PCIe clock adjustments in relation to the GPU board, or other stalls in the PCIe transfer path, for example the location of switches with respect to the GPU and CPU socket, and of course the operating system jitter. The same has been observed when the paged memory transfers were replaced with pinned memory transfers. A consistent feature in the host to device transfers on K20 is a high spike at the first iteration, which is missing in the K80 measurements. Although not important, it shows a difference between the PCIe 2.0 and 3.0 behaviour. The overall spread including data transfers is $\sim 1\%$, and the outliers minimal, but studies suggested [133] that this is an upper limit for the desired overall jitter for the total sampling time in order to preserve full imaging performance, and the solver is only part of the operations that contribute to the control loop pipeline. Furthermore, the spread between two extremes in time units is over 15 $\mu s$, which is the 1% of the 2 ms sampling time assumed in this study. It is also meaningful to investigate the sources in this simple case, to characterize some



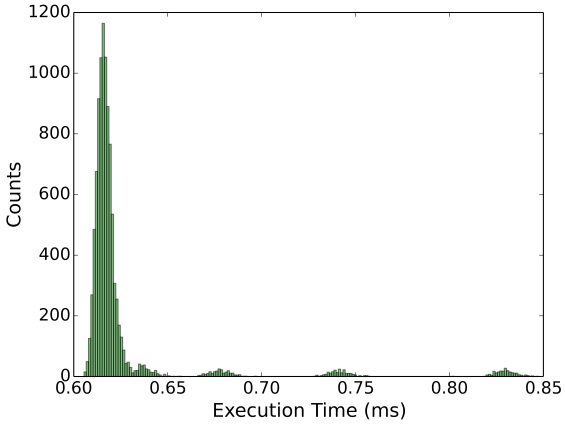

(a) Host to device transfers on K20



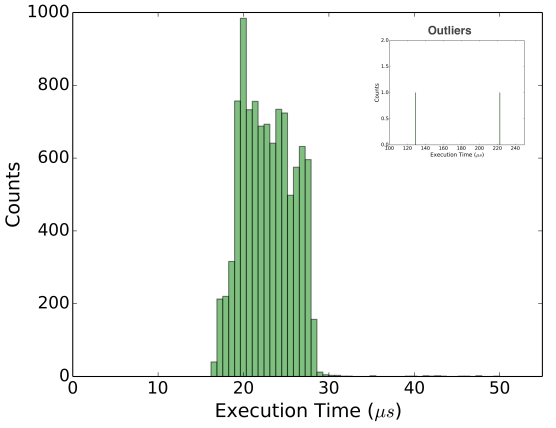

(b) Host to device transfers on K80



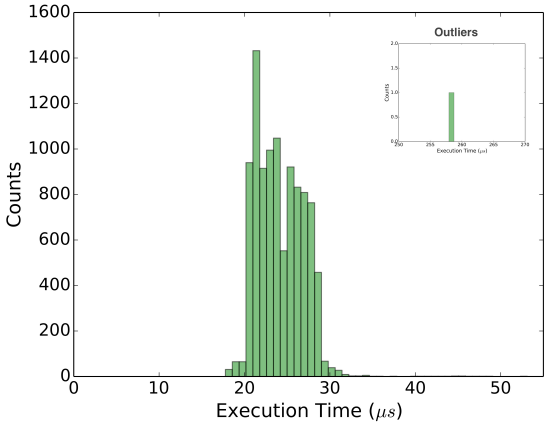

(c) SGEMV on K20




(d) SGEMV on K80

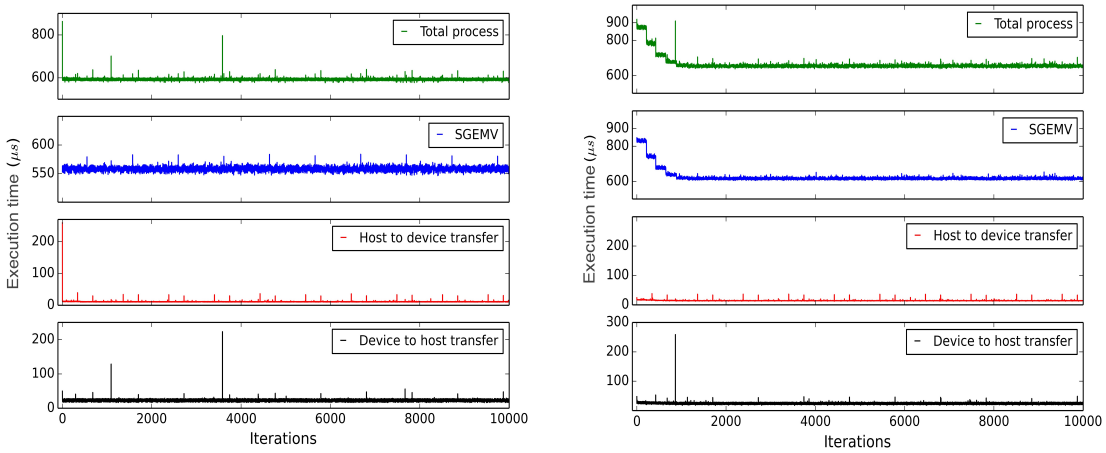



(e) Device to host transfers on K20




(f) Device to host transfers on K80

Figure 6.7: Jitter histograms for SGEMV, Host to device transfers and device to host transfers on a K20 and K80 GPU.




(a) execution times for 10000 iterations on K20

(b) execution times for 10000 iterations on K80

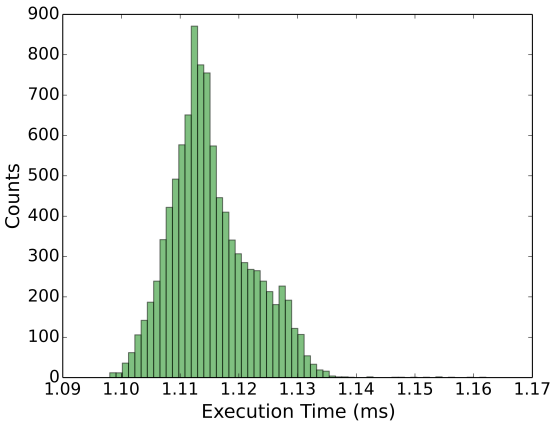
Figure 6.8: Execution times patterns during 10000 iterations for the full process and each of Host to device transfers, SGEMV, and Device to host transfers separately on a K20 and K80 GPU.

basic features of the operating system behaviour with graphics cards computational operations, and look at ways that this can be further minimized. Going back to the data transfers considerations, if the transfers are done asynchronously using CUDA streams, we see that the jitter due to the memory copies is effectively hidden, along with the whole transfers. The overlapping behaviour can be visualised by running the application through the NVIDIA Visual Profiler (figure 6.9). The histogram of the total execution times when using overlapped memory transfers and SGEMV executions now appears to have a Gaussian distribution with $\sim 0.63\%$ standard deviation. It should be noted that the measurements were taken over 2 iterations at a time, in order to accommodate the overlap, but the result shows still shorter outliers and the spread in percentage is also smaller.

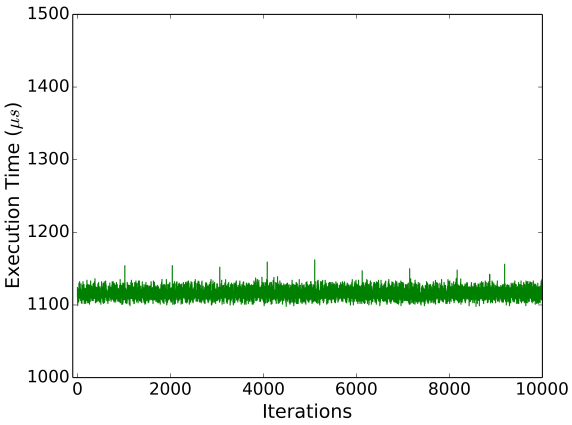
On the K80, figures 6.7d and 6.8b show a large gradual variation in the execution of the routine on the GPU, that takes more than 1000 iterations to settle. The line plot (6.8b) shows clearly that there is a gradual speedup in execution. This points to the possibility that the driver is adjusting the device hardware in some way. Indeed,


(a) Overlapping GPU kernel and memory transfers


(b) Non-overlapping GPU kernel and memory transfers



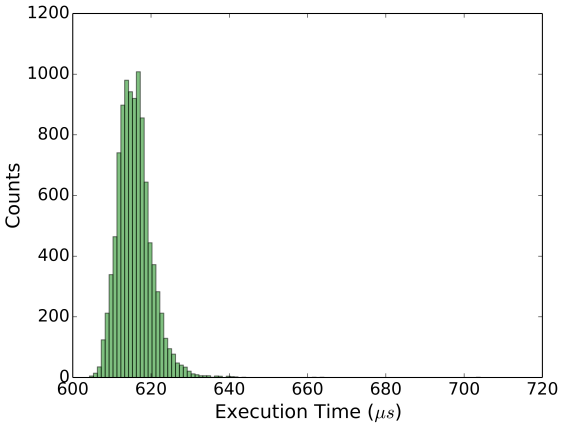

(c) Total process distribution of execution times



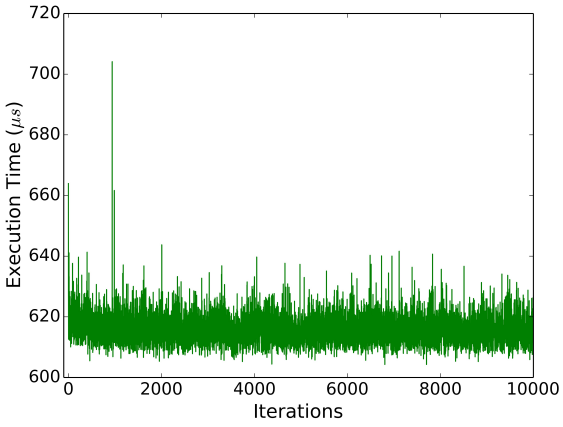

(d) Total process execution pattern for 10000 iterations

Figure 6.9: Overlapping behaviour on a K20 GPU.

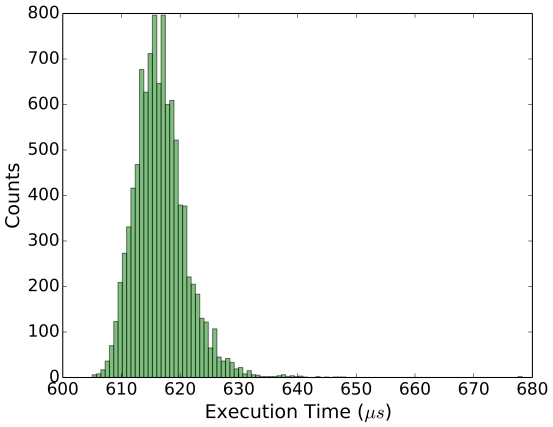
K80 has a new driver feature called 'auto boost', which adjusts the processor graphics clock automatically according to the application, and is always enabled by default, but the user can change the settings using the 'nvidia-smi' driver tool. Auto boost is generally beneficial, because it can boost the clock at a value higher than the preset limit that a user can set, but care must be taken to keep the clock setting at the maximum to avoid load balancing issues from the difference between the initial and peak frequencies [146]. To see whether this would help, another set of tests was run with the clock setting to maximum and auto boost enabled and disabled to assess which is the most stable. The results are shown in figure 6.10 and we can see that the issue is mostly resolved in both cases, with the mean and standard deviation being approximately the same. There still is a delay in speed increase which takes several iterations to settle, and increases the spread towards the tail, compared to the K20 measurements.



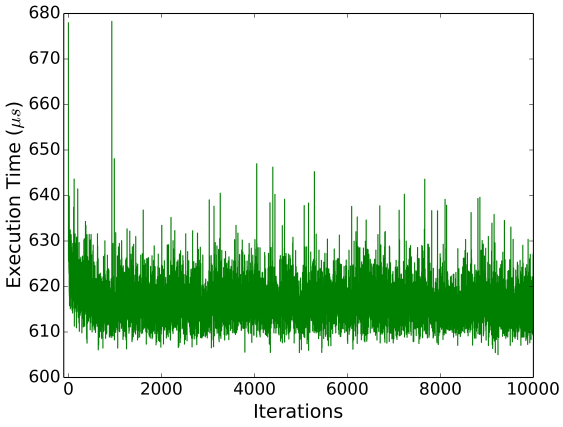

(a) SGEMV distribution with autoboost enabled and maximum clock




(b) SGEMV timeline with autoboost enabled and maximum clock




(c) SGEMV distribution with autoboost disabled and maximum clock



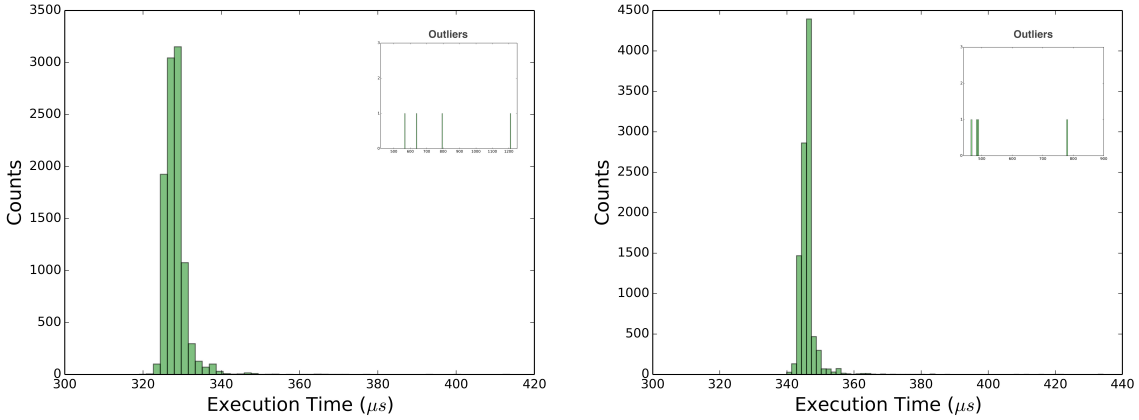

(d) SGEMV timeline with autoboost disabled and maximum clock

Figure 6.10: Autoboost behaviour on a K80 GPU.

### Multiple GPUs

In the case of multiple GPUs we provide the total process distribution for each pair of K20 and K80 cards, as it is generally accepted that in multi-GPU operation it is appropriate to use cards of the same specifications. As mentioned before, combining different cards is not recommended in practical applications for two main reasons. First, because the different hardware characteristics restrict the performance of the most powerful of the cards, as there will be an implicit synchronisation delay in order to get a complete result. This will align the execution time with the slowest

device. Second, the different hardware specifications imply incompatibilities that can compromize stability in performance, and exclude the use of newer or specific hardware and software features. Histograms for each pair of K80 and K20 cards, are shown in 6.11. The behaviour does not change significantly, and there is no additional jitter observed.




(a) 2 × K20 (b) 2 × K80

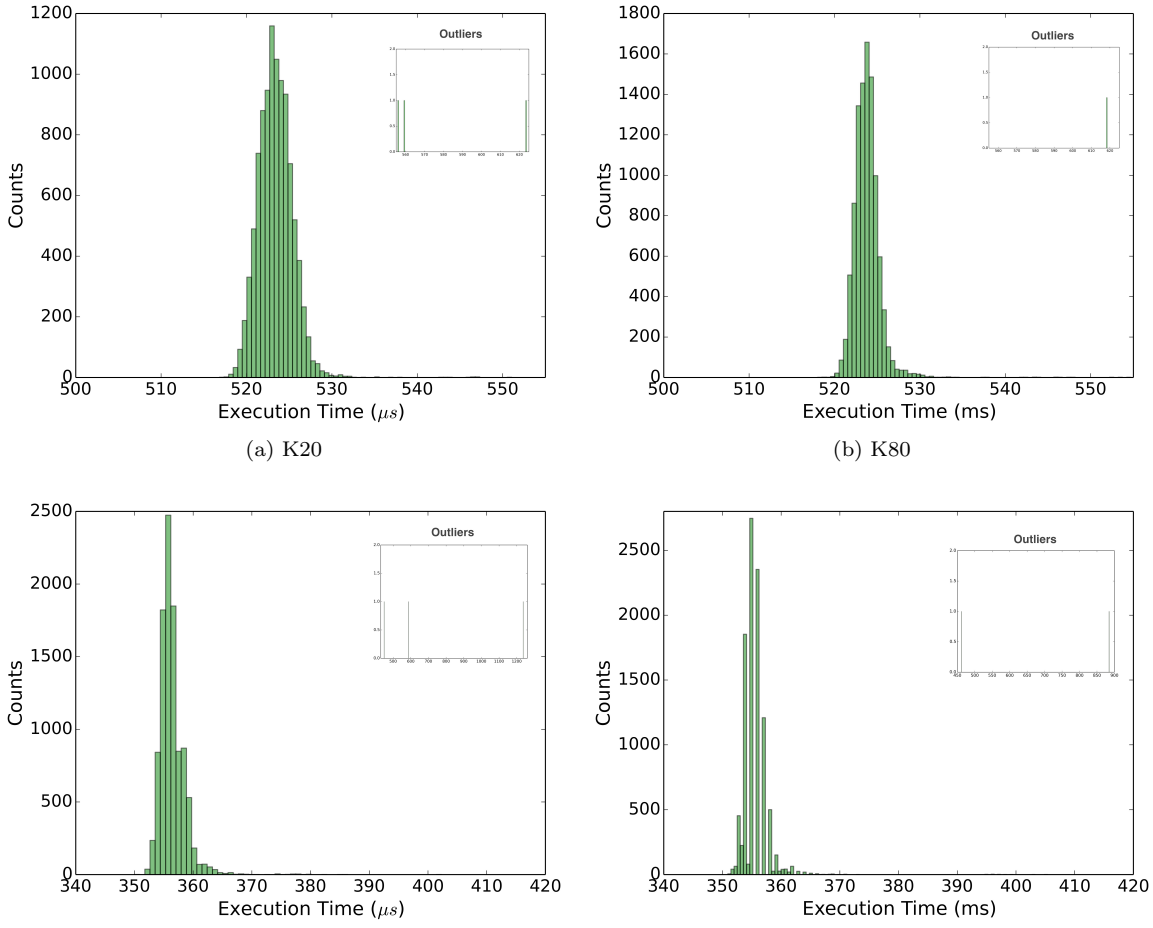
Figure 6.11: Jitter histograms for dense matrix vector multiplications using (a) 2 × K20 and (b) 2 × K80.

#### 6.2.3.2 Sparse Matrix-vector multiplication

The behaviour of the cuSPARSE matrix-vector multiplication routine did not show any difference to SGEMV. Using a 7701 × 7701 sparse matrix which was used in the simulation to apply conjugate gradients on the MCAO system, we get the distributions of figure 6.12. We present only the result of the SPMV operation, as the behaviour for data transfers has been examined in the previous sections.

#### 6.2.3.3 Conjugate Gradients

The conjugate gradients routine applies the following operations: 1. transfer of gradient vector to GPU via PCIe, 2. perform CSRMV, vector COPY and DOT




(a) K20

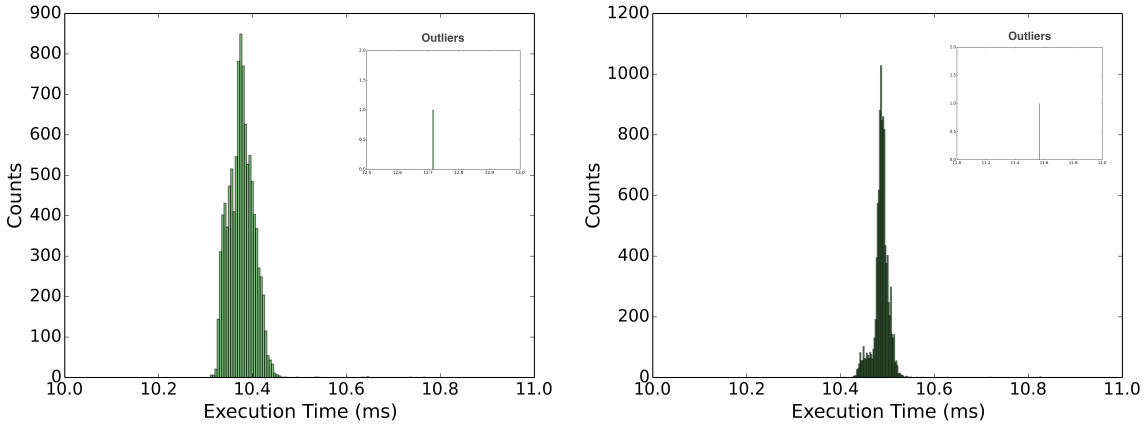
(b) K80

(c) 2 × K20

(d) 2 × K80

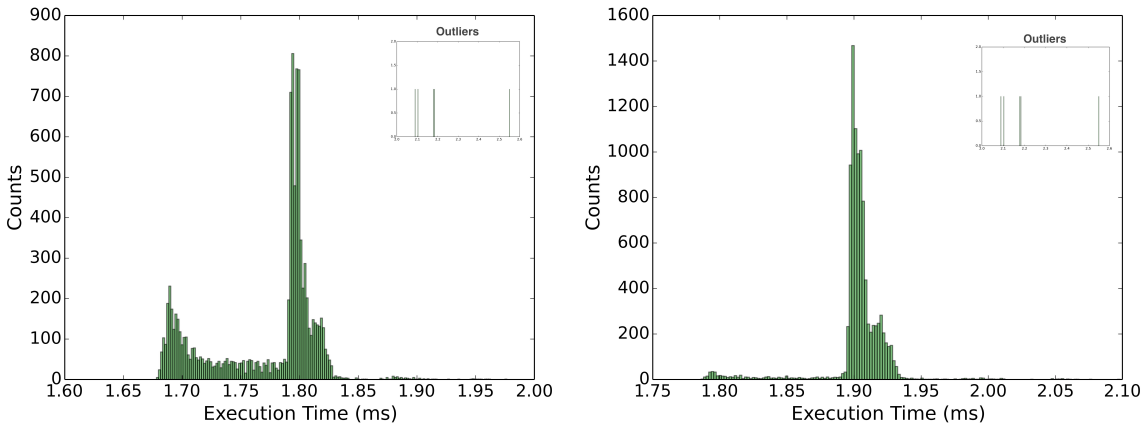
Figure 6.12: Jitter histograms for sparse matrix vector multiplications using (a) 1 × K20, (b) 1 × K80, (c) 2 × K20, (d) 2 × K80.

product for initialization, 3. perform 15 iterations of SCAL (scalar multiplication), AXPY (vector addition), CSRMV, 2 × DOT, 2 × AXPY, DOT, with the first 2 operations omitted in the first iteration, 4. transfer command vector back to the CPU. Timings of the whole process were taken over 10000 runs, initially using the matrix created in the simulation. The distribution of results using a single K20 and a single K80 are shown in figure 6.13. We then increased the sparsity of the matrix to bring computation time to the 1ms region (fig. 6.14). These distributions show the effect of launching consecutive GPU kernels from the host. Each kernel launch from the host incurs additional latency from the host that varies with the system




(a) K20 (b) K80

Figure 6.13: Jitter histograms for 15 Conjugate Gradients iterations using (a) $1 \times$K20, (b) $1 \times$K80.




(a) K20 (b) K80

Figure 6.14: Jitter histograms for 15 Conjugate Gradients iterations using a very sparse matrix. (a) $1 \times$ K20, (b) $1 \times$ K80.

jitter. The spread increases to 2.67% for K20, and it is less on K80, 1.35%, but still higher than what we've seen for a single operation.

## 6.3 Discussion

The conjugate gradients offer some interesting opportunities for trade-offs between image quality and speed. The imaging quality achieved by the simulation is unaffected by a matrix configuration that allows a 15% reduction in execution time,

and in particular the ensquared energy is hardly affected by any of the sparsity and iteration variations, which could reduce the time up to 65%, if such a configuration per choice of observation were applicable (for example in the ideal case were the real-time goal is achieved, one could perform a reconstruction with more layers, increasing the imaging quality). We find however that there is a limit to the execution time we can achieve with conjugate gradients using a general matrix.

In terms of jitter, we see that for matrix-vector operations the GPUs have very good performance, with the execution latency varying by up to $\sim 20\mu s$, and that this is further improved as jitter is hidden between PCIe transfers and single GPU operation launch, which shows the importance of overlapping for low latency and real-time operations. We also see how different device features can have a significant effect, such as the very long delay in the execution of the first 1000 frames on a K80 card, and that these need to be considered and improved if possible. For the Conjugate Gradients, the jitter measurements show that the repeated launching of GPU functions on the host increases the jitter. The significance of jitter increases with shorter latencies, and this shows that there is a need for tuning the operating system to match the latency and jitter requirements of the application. The system on test uses an Ubuntu low latency kernel (v3.13) with the default settings, which is widely known to not be optimal for computational applications, and would need to be tuned to match an application. For this reason, CPU jitter measurements were not considered for comparison.

Latency and jitter for a CPU-only solution have been tested and reported for the TMT Narrow Field Infrared Adaptive Optics System (NFIRAOS) RTC trade

studies in [88, 147–149], and it has been found that the jitter for partial end-to-end real-time control computations on one machine was up to 110 $\mu s$ from the mean in the worst case, which was seen as an acceptable margin for the 1.25 $ms$ frame time specified for NFIRAOS. There are however a few things to note here. First, the jitter margin and applicability of such a method depend on the instrument specifications. The NFIRAOS system will have six LGS WFS with $60 \times 60$ sub-apertures each, resulting in a total of 31008 sub-aperture gradients, and a command vector of 7673 elements to be applied to two DMs. The system will be processed by 7 CPU servers, 6 for the pixel processing and MVM, and one for the DM commands summing. The MAORY system considered in this thesis, will have at least 9 high order WFS with up to $80 \times 80$ sub-apertures each, amounting to over 90000 sub-aperture slopes with a command vector of over 7000 elements, so it would require more servers to perform the operation, and both latency and jitter could increase due to communications and synchronisation overhead, or the amount of work assigned to each server. Second, the benchmarks performed in these studies measured the jitter on the roundtrip of the data on only one machine that processes 1/2 WFS. The roundtrip is defined from sending the first WFS pixel to the server, until the 2 DM command command vectors produced by each CPU on the server are received via ethernet and summed. The system is highly tuned in all aspects, including ethernet transfers, it is not clear however what additional overhead could be expected from using all 6 servers. A third point that is very important is that in these trade studies, the jitter requirement is not yet determined, and the acceptability of the result is based on the requirement for no missed frames and the worst case latency observed.

It is not clear if the effect of latency and jitter in image performance has or is being measured or estimated. Finally, the studies also considered GPUs, and they point out that they show indeed lower latency and jitter, as well as lower energy and space consumption by needing less PC units, and also better maintainability and lower cost, but CPUs were preferred because they require less programming effort, are more homogenous with less complexity of operation, mainstream devices that are most likely to be backwards compatible and to provide support for real-time operating systems. Therefore the choice of architecture further depends on a trade-off between the operational requirements of the instrument, provided that it fulfils the technical goals first. The fulfilment of the technical goals is something that is also instrument depended and, in the case of latency jitter, there is still a need to profile the effects in order to determine a specification.

Overall, it is clear from both the performance and jitter results that the CG has potential for speedup on GPUs, but solutions that apply structured matrices should be sought, which would enable all the computations to happen during one kernel launch, thus avoiding both extra jitter and time spent in GPU memory transfers. A structured matrix that allows massive parallelism would be preferable for the use of GPUs. CPU solutions considered are showing much worse performance in terms of jitter, but they can be acceptable and comparable depending on the instrument goals and requirements.

# Chapter 7

# Conclusions

This chapter summarises my findings and conclusions from the study, and considers future research that this work could potentially inspire.

## 7.1 Summary

There are two main approaches in solving the problem of atmospheric tomography within the accuracy and timing requirements set by Adaptive Optics Instruments for ELTs. The first approach is algorithmic, aiming at providing less data intensive data structures by exploiting mathematical properties of the problem, which are often model-based. The second approach is to make use of the latest available device architectures with potential for very high data parallelism and solve the problem by only applying parallel implementation techniques. The key to achieving shorter execution time in large computational applications today is parallelism, and as such, besides reducing the volume of the data to be processed, one must also seek the potential for parallel execution when developing solutions. A matrix-vector

multiplication with a pre-computed inverse is highly parallelisable, but the volume of the data to be processed for producing the inverse, and the size of it in some adaptive optics configurations are still prohibitive. In addition, if the matrix has to be updated at set time intervals, it may not be possible to compute it timely.

The most researched advanced computational algorithms were reviewed, and it was found that until now, although there has been extensive work and interesting ideas that have proven results, none of them has achieved the goals of tomographic reconstruction in real-time entirely. A simple conjugate gradient approach has the potential to avoid direct matrix inversion, but it needs a sparse formulation and fast convergence, something that motivates research for suitable tomography models and pre-conditioners. Its recursive nature creates a difficulty for a parallel implementation. The FD-PCG has very good convergence, but it has been found to have less imaging performance. In addition, the Fourier operations and indirect memory accesses in multiple steps can create a bottleneck in parallel architectures. It is however parallelized at the atmospheric layer level by providing block structures. The FrIM and CuReD have both shown optimal performance, but are still immature in developing a tomographic solution. Time domain solutions are superior in imaging quality, but they are even more computationally expensive.

Plain CG with an appropriate approximation of the phase covariance matrix achieved a reasonable imaging performance on simulation for a partial tomographic reconstruction modelled after the MAORY MCAO system, whose computational performance limit was examined with technical means, using the best hardware configuration and the largest allowable sparsity for a general matrix. The limitation

was found to be the matrix structure, which verifies the need for using different mathematical concepts, but aiming towards parallelism more than data volume reduction.

GPUs demonstrated a progressive advancement in technology, and have become a highly reliable architecture for future use. In terms of real-time control operation, the main bottleneck is the additional data transfers needed through the PCIe bus, something that also contributes to system jitter, which, as we've shown can be overcome by simply making use of the API features, and in general the measurements showed that the jitter is very low. In addition, the NVIDIA GPU technology roadmap includes a new interconnect technology, NVLink, which is expected to increase the bandwidth between the host and device dramatically [115]. The bandwidth is also likely to be increased on the device with the development of stacked 3-D memory technology for the next generation Pascal.

I have shown that system jitter related to GPU operations can have a significant effect, particularly in the case of multiple sequential operations such as CG has, even though it was demonstrated to be much lower than when using parallelism in multi-core CPUs. In this context, operating system and device driver tuning is essential and may help mitigate the effect.

A CG algorithm based on library functions is inherently problematic in terms of jitter, and in terms of total execution speed when done with general matrix format. Multiple device memory transfers is a significant drawback which makes the method bandwidth limited on the device, in addition to the fact that it is not directly parallelizable, therefore with the current technology, it appears that the

best solution would be a structured matrix format that will enable the execution of all iterations in a single kernel, using on-chip resources. The release of hardware with stacked memory is expected to increase device bandwidth significantly, with a potential to greatly reduce the delays of the device memory transfers. However, the performance and behaviour of this new technology with various algorithms is yet unknown.

## 7.2 Future work

Based on my conclusions, I believe that more work in simulating more accurately partial frame delay and jitter effects on imaging would be advantageous.

Tuning the operating system requires specialized knowledge of engineering and computer and low level operating system architecture. Since the systems are specifically designed, and in many parts they may contain customized interfaces, communication protocols, processing algorithms and special purpose hardware, targeted research on system tuning should be incorporated in the development of E-ELT systems.

Finally, research on tomography algorithms should also become more focused on the capacity for parallelism, since today's technology and particularly GPU's provide the opportunity to exploit extremely high processing power.

# Bibliography


[1] H. W. Babcock. "The Possibility of Compensating Astronomical Seeing". *Publications of the Astronomical Society of the Pacific*, 65:229 (1953).

[2] C. R. Vogel. *Computational Methods for Inverse Problems*. SIAM (2002).

[3] B. L. Roux, C. Kulcsár, L. M. Mugnier, et al. "Optimal control law for classical and multiconjugate adaptive optics". *J. Opt. Soc. Am. A*, 21(7):1261–1276 (2004).

[4] R. E. Hufnagel. "Variations of atmospheric turbulence". In *Optical Propagation through Turbulence*, OSA Technical Digest Series. Optical Society of America (1974).

[5] G. C. Valley. "Isoplanatic degradation of tilt correction and short-term imaging systems". *Appl. Opt.*, 19(4):574–577 (1980).

[6] P. B. Ulrich. "Hufnagel-Valley profiles for specified values of the coherence length and isoplanatic angle". Technical Report MA-TN-88-013, W.J. Schafer Associates (1988).

[7] J. Vernin, A. Agabi, R. Avila, et al. "1998 Gemini Site Testing Campaign, Cerro Pachon and Cerro Tololo". Technical report, International Gemini Project Office (2000).

[8] A. N. Kolmogorov. "Dissipation of Energy in the Locally Isotropic Turbulence". *Royal Society of London Proceedings Series A*, 434:15–17 (1991).

[9] V. I. Tatarskii. *Wave propagation in a turbulent medium*. McGraw-Hill (1961).

[10] T. von Karman. "Progress in the Statistical Theory of Turbulence". *Proceedings of the National Academy of Science*, 34:530–539 (1948).

[11] R. K. Tyson. *Adaptive Optics Engineering Handbook*. CRC Press (1999).

[12] D. L. Fried. "Statistics of a Geometric Representation of Wavefront Distortion". *J. Opt. Soc. Am.*, 55(11):1427–1431 (1965).

[13] J. W. Hardy. *Adaptive Optics for Astronomical Telescopes*. Oxford University Press (1998).

[14] F. Roddier. *Adaptive Optics in Astronomy*. Cambridge University Press (1999).

[15] F. Zernike. "Diffraction theory of the knife-edge test and its improved form, the phase-contrast method". , 94:377–384 (1934).

[16] R. J. Noll. "Zernike polynomials and atmospheric turbulence". *J. Opt. Soc. Am.*, 66(3):207–211 (1976).

[17] R. M. Myers, A. J. Longmore, C. R. Benn, et al. "NAOMI adaptive optics system for the 4.2m William Herschel telescope". In P. L. Wizinowich and D. Bonaccini, editors, *Adaptive Optical System Technologies II*, volume 4839 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 647–658 (2003).

[18] R. K. Tyson. *Principles of Adaptive Optics*. CRC Press, third edition edition (2011).

[19] M. A. van Dam, D. L. Mignant, and B. A. Macintosh. "Performance of the Keck Observatory adaptive-optics system". *Appl. Opt.*, 43(29):5458–5467 (2004).

[20] G. Rousset, F. Lacombe, P. Puget, et al. "NAOS, the first AO system of the VLT: on-sky performance". In P. L. Wizinowich and D. Bonaccini, editors, *Adaptive Optical System Technologies II*, volume 4839 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 140–149 (2003).

[21] P.-Y. Madec. "Overview of deformable mirror technologies for adaptive optics and astronomy". In *Adaptive Optics Systems III*, volume 8447 of *Proc. SPIE*, pages 844705–844705–18 (2012).

[22] R. K. Tyson and B. W. Frazier. *Field Guide to Adaptive Optics*, volume FG03 of *SPIE Field Guides*. SPIE Press (2004).

[23] D. L. Fried. "Least-square fitting a wave-front distortion estimate to an array of phase-difference measurements". *J. Opt. Soc. Am.*, 67(3):370–375 (1977).

[24] D. T. Gavel. "Suppressing anomalous localized waffle behavior in least-squares wavefront reconstructors". In *Adaptive Optical System Technologies II*, volume 4839 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 972–980 (2003).

[25] E. P. Wallner. "Optimal wave-front correction using slope measurements". *J. Opt. Soc. Am.*, 73(12):1771–1776 (1983).

[26] B. L. Ellerbroek. "Efficient computation of minimum-variance wave-front reconstructors with sparse matrix techniques". *J. Opt. Soc. Am. A*, 19(9):1803–1816 (2002).

[27] T. Fusco, J.-M. Conan, G. Rousset, et al. "Optimal wave-front reconstruction strategies for multiconjugate adaptive optics". *J. Opt. Soc. Am. A*, 18(10):2527–2538 (2001).

[28] J. M. Beckers. "Increasing the Size of the Isoplanatic Patch with Multiconjugate Adaptive Optics". In M.-H. Ulrich, editor, *European Southern Observatory Conference and Workshop Proceedings*, volume 30 of *European Southern Observatory Conference and Workshop Proceedings*, page 693 (1988).

[29] D. L. Fried. "Anisoplanatism in adaptive optics". *J. Opt. Soc. Am.*, 72(1):52–52 (1982).

[30] D. L. Fried and J. F. Belsher. "Analysis of fundamental limits to artificial-guide-star adaptive-optics-system performance for astronomical imaging". *J. Opt. Soc. Am. A*, 11(1):277–287 (1994).

[31] G. A. Tyler. "Rapid evaluation of d0: the effective diameter of a laser-guide-star adaptive-optics system". *J. Opt. Soc. Am. A*, 11(1):325–338 (1994).

[32] R. Foy and A. Labeyrie. "Feasibility of adaptive telescope with laser probe". , 152:L29–L31 (1985).

[33] M. Tallon and R. Foy. "Adaptive telescope with laser probe - Isoplanatism and cone effect". , 235:549–557 (1990).

[34] S. Esposito. "Introduction to Multi-Conjugate Adaptive Optics systems". *Comptes Rendus Physique*, 6:1039–1048 (2005).

[35] R. Ramlau and M. Rosensteiner. "An efficient solution to the atmospheric turbulence tomography problem using Kaczmarz iteration". *Inverse Problems*, 28(9):095004 (2012).

[36] S. K. Ramsay, M. M. Casali, J. C. González, et al. "The E-ELT instrument roadmap: a status report". In *Ground-based and Airborne Instrumentation for Astronomy V*, volume 9147, pages 91471Z–91471Z–10 (2014).

[37] D. C. Johnston and B. M. Welsh. "Analysis of multiconjugate adaptive optics". *J. Opt. Soc. Am. A*, 11(1):394–408 (1994).

[38] R. Davies and M. Kasper. "Adaptive Optics for Astronomy". , 50:305–351 (2012).

[39] F. Hammer, F. Sayède, E. Gendron, et al. "The FALCON Concept: Multi-Object Spectroscopy Combined with MCAO in Near-IR". In J. Bergeron and G. Monnet, editors, *Scientific Drivers for ESO Future VLT/VLTI Instrumentation*, page 139 (2002).

[40] National E-ELT instrumentation consortium website (Netherlands). "MOSAIC: The multi-object spectrograph on the E-ELT". `http://www.e-elt.nl/index.php?v=mosaic` (2016).

[41] S. Morris and J.-G. Cuby. "EAGLE: An Adaptive Optics Fed, Multiple Integral Field Unit, Near-infrared Spectrograph". *The Messenger*, 140:22–23 (2010).

[42] M. C. Britton. "Numerical simulations of single conjugate adaptive optics systems". In *Astronomical Telescopes and Instrumentation*, pages 609–616. International Society for Optics and Photonics (2004).

[43] J. Bardsley. "Wavefront Reconstruction Methods for Adaptive Optics Systems on Ground-Based Telescopes". *SIAM Journal on Matrix Analysis and Applications*, 30(1):67–83 (2008).

[44] C. R. Vogel. "Sparse matrix methods for wavefront reconstruction, revisited". In *Advancements in Adaptive Optics*, volume 5490 of *Proc. SPIE*, pages 1327–1335 (2004).

[45] L. Gilles, C. R. Vogel, and B. L. Ellerbroek. "Multigrid preconditioned conjugate-gradient method for large-scale wave-front reconstruction". *J. Opt. Soc. Am. A*, 19(9):1817–1822 (2002).

[46] C. R. Vogel and Q. Yang. "Multigrid algorithm for least-squares wavefront reconstruction". *Appl. Opt.*, 45(4):705–715 (2006).

[47] Q. Yang, C. R. Vogel, and B. L. Ellerbroek. "Fourier domain preconditioned conjugate gradient algorithm for atmospheric tomography". *Appl. Opt.*, 45(21):5281–5293 (2006).

[48] C. R. Vogel and Q. Yang. "Fast optimal wavefront reconstruction for multiconjugate adaptive optics using the Fourier domain preconditioned conjugate gradient algorithm". *Opt. Express*, 14(17):7487–7498 (2006).

[49] E. Thiébaut and M. Tallon. "Fast minimum variance wavefront reconstruction for extremely large telescopes". *J. Opt. Soc. Am. A*, 27(5):1046–1059 (2010).

[50] Thirty Meter Telescope. "NFIRAOS RTC Algorithm Description". Technical Report TMT.AO.DRD.08.002.REL05, Thirty Meter Telescope (2011).

[51] R. G. Lane, A. Glindemann, and J. C. Dainty. "Simulation of a Kolmogorov phase screen". *Waves in Random Media*, 2(3):209–224 (1992).

[52] C. Kulcsár, H.-F. Raynaud, C. Petit, et al. "Optimal control, observers and integrators in adaptive optics". *Optics express*, 14(17):7464–7476 (2006).

[53] P. Massioni, C. Kulcsár, H.-F. Raynaud, et al. "Fast computation of an optimal controller for large-scale adaptive optics". *J. Opt. Soc. Am. A*, 28(11):2298–2309 (2011).

[54] C. Petit, J.-M. Conan, C. Kulcsár, et al. "Linear quadratic Gaussian control for adaptive optics and multiconjugate adaptive optics: experimental and numerical analysis". *J. Opt. Soc. Am. A*, 26(6):1307–1325 (2009).

[55] A. Costille, C. Petit, J.-M. Conan, et al. "Wide field adaptive optics laboratory demonstration with closed-loop tomographic control". *J. Opt. Soc. Am. A*, 27(3):469–483 (2010).

[56] G. Sivo, C. Kulcsár, J.-M. Conan, et al. "First on-sky SCAO validation of full LQG control with vibration mitigation on the CANARY pathfinder". *Opt. Express*, 22(19):23565–23591 (2014).

[57] H. W. Sorenson. "Least-squares estimation: from Gauss to Kalman". *IEEE Spectrum*, 7:63–68 (1970).

[58] C. Correia, J.-M. Conan, C. Kulcsár, et al. "Adapting optimal LQG methods to ELT-sized AO systems". In *Adaptative Optics for Extremely Large Telescopes*, volume 1, page 7003 (2010).

[59] M. Yudytskiy, T. Helin, and R. Ramlau. "Finite element-wavelet hybrid algorithm for atmospheric tomography". *J. Opt. Soc. Am. A*, 31(3):550–560 (2014).

[60] T. Helin and M. Yudytskiy. "Wavelet methods in multi-conjugate adaptive optics". *Inverse Problems*, 29(8):085003 (2013).

[61] N. Bharmal, U. Bitenc, A. Basden, et al. "A hierarchical wavefront reconstruction algorithm for gradient sensors". In S. Esposito and L. Fini, editors, *Proceedings of the Third AO4ELT Conference*, page 48 (2013).

[62] M. Rosensteiner. "Wavefront reconstruction for extremely large telescopes via CuRe with domain decomposition". *J. Opt. Soc. Am. A*, 29(11):2328–2336 (2012).

[63] U. Bitenc, M. Rosensteiner, N. Bharmal, et al. "Tests of novel wavefront reconstructors on sky with CANARY". In S. Esposito and L. Fini, editors, *Proceedings of the Third AO4ELT Conference*, page 42 (2013).

[64] J. R. Shewchuk. "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain". Technical report, Carnegie Mellon University, Pittsburgh, PA, USA (1994).

[65] L. A. Poyneer, D. T. Gavel, and J. M. Brase. "Fast wave-front reconstruction in large adaptive optics systems with use of the Fourier transform". *J. Opt. Soc. Am. A*, 19(10):2100–2111 (2002).

[66] R. Bracewell. *Fourier Analysis and Imaging*. Springer US (2003).

[67] M. Roggemann, B. Welsh, and B. Hunt. *Imaging Through Turbulence*. CRC Press laser and optical science and technology series. Taylor & Francis (1996).

[68] M. Tallon, E. Thiébaut, and C. Béchet. "A Fractal Iterative Method for Fast Wavefront Reconstruction for Extremely Large Telescopes". In *Adaptive Optics: Analysis and Methods/Computational Optical Sensing and Imaging/Information Photonics/Signal Recovery and Synthesis Topical Meetings on CD-ROM*, page PMA2. Optical Society of America (2007).

[69] A. Fournier, D. Fussell, and L. Carpenter. "Computer Rendering of Stochastic Models". *Commun. ACM*, 25(6):371–384 (1982).

[70] B. B. Mandelbrot and B. Dwyer. "Technical Correspondence: Comment on Computer Rendering of Fractal Stochastic Models". *Commun. ACM*, 25(8):581–583 (1982).

[71] G. S. P. Miller. "The Definition and Rendering of Terrain Maps". *SIGGRAPH Comput. Graph.*, 20(4):39–48 (1986).

[72] M. Tallon, C. Béchet, I. Tallon-Bosc, et al. "Performance of MCAO on the E-ELT using the Fractal Iterative Method for fast atmospheric tomography". In *Second International Conference on Adaptive Optics for Extremely Large Telescopes.*, page 63 (2011).

[73] L. Feng, E. Fedrigo, C. Béchet, et al. "Computational performance comparison of wavefront reconstruction algorithms for the European Extremely Large Telescope on multi-CPU architecture". *Applied optics*, 51(16):3564–3583 (2012).

[74] M. Tallon, I. Tallon-Bosc, C. Béchet, et al. "Fractal iterative method for fast atmospheric tomography on extremely large telescopes" (2010).

[75] R. N. Paschall and D. J. Anderson. "Linear quadratic Gaussian control of a deformable mirror adaptive optics system with time-delayed measurements". *Appl. Opt.*, 32(31):6347–6358 (1993).

[76] D. T. Gavel and D. Wiberg. "Toward Strehl-optimizing adaptive optics controllers" (2003).

[77] L. A. Poyneer, B. A. Macintosh, and J.-P. Véran. "Fourier transform wavefront control with adaptive prediction of the atmosphere". *J. Opt. Soc. Am. A*, 24(9):2645–2660 (2007).

[78] M. Gray and B. Le Roux. "Utilization of the Ensemble Kalman Filter: an optimal control law for the adaptive optics of the E-ELT". In S. Boissier, M. Heydari-Malayeri, R. Samadi, et al., editors, *SF2A-2010: Proceedings of the Annual meeting of the French Society of Astronomy and Astrophysics*, page 73 (2010).

[79] M. Gray, C. Petit, S. Rodionov, et al. "Local ensemble transform Kalman filter, a fast non-stationary control law for adaptive optics on ELTs: theoretical aspects and first simulation results". *Opt. Express*, 22(17):20894–20913 (2014).

[80] M. Gray and B. Le Roux. "Ensemble Transform Kalman Filter, a nonstationary control law for complex AO systems on ELTs: theoretical aspects and first simulations results" (2012).

[81] M. Gray, C. Petit, S. Rodionov, et al. "Local Ensemble Transform Kalman Filter: a non stationary control law for complex adaptive optics systems on ELTs". In S. Esposito and L. Fini, editors, *Proceedings of the Third AO4ELT Conference*. INAF - Osservatorio Astrofisico di Arcetri, Firenze (2013).

[82] M. Zhariy, A. Neubauer, M. Rosensteiner, et al. "Cumulative wavefront reconstructor for the Shack-Hartmann sensor". *Inverse Problems and Imaging*, 5(4):893–913 (2011).

[83] M. Rosensteiner. "Cumulative Reconstructor: fast wavefront reconstruction algorithm for Extremely Large Telescopes". *J. Opt. Soc. Am. A*, 28(10):2132–2138 (2011).

[84] U. Bitenc, M. Rosensteiner, N. Bharmal, et al. "Tests of novel wavefront reconstructors on sky with CANARY". In S. Esposito and L. Fini, editors, *Proceedings of the Third AO4ELT Conference*, page 42 (2013).

[85] U. Bitenc, A. Basden, N. A. Bharmal, et al. "On-sky tests of the CuReD and HWR fast wavefront reconstruction algorithms with CANARY". *Monthly Notices of the Royal Astronomical Society*, 448(2):1199–1205 (2015).

[86] R. H. Hudgin. "Wave-front reconstruction for compensated imaging". *J. Opt. Soc. Am.*, 67(3):375–378 (1977).

[87] M. Rosensteiner and R. Ramlau. "Kaczmarz algorithm for multiconjugated adaptive optics with laser guide stars". *J. Opt. Soc. Am. A*, 30(8):1680–1686 (2013).

[88] J.-P. Véran, C. Boyer, B. L. Ellerbroek, et al. "Results of the NFIRAOS RTC trade study" (2014).

[89] L. Wang. "Design and Testing of GPU based RTC for TMT NFIRAOS". In S. Esposito and L. Fini, editors, *Proceedings of the Third AO4ELT Conference*, page 17 (2013).

[90] V. Volkov and J. W. Demmel. "Benchmarking GPUs to tune dense linear algebra". In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 31. IEEE Press (2008).

[91] NVIDIA. "Intel Xeon Phi vs. NVIDIA Tesla GPU: Just the Facts". `http://www.nvidia.com/object/justthefacts.html` (2013).

[92] NVIDIA. "Tesla K40 GPU Accelerator - Board Specification" (2013).

[93] NVIDIA. "Tesla K80 GPU Accelerator - Board Specification" (2014).

[94] GPGPU.org. "General-Purpose Computation on Graphics Hardware - GPGPU Developer Resources". `http://gpgpu.org/developer` ().

[95] U. J. Kapasi, S. Rixner, W. J. Dally, et al. "Programmable stream processors". *Computer*, 36(8):54–62 (2003).

[96] M. Pharr and R. Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional (2005).

[97] NVIDIA. "NVIDIA CUDA Zone". https://developer.nvidia.com/cuda-zone ().

[98] AMD. "The AMD OpenCL Zone". http://developer.amd.com/tools-and-sdks/opencl-zone (2015).

[99] NVIDIA. "NVIDIA GeForce 8800 GPU Architecture Overview - Technical Brief" (2006).

[100] Microsoft . "DirectX Graphics and Gaming". https://msdn.microsoft.com/en-gb/library/windows/desktop/ee663274.aspx (2015).

[101] NVIDIA. "NVIDIA GeForce GTX 200 GPU Architectural Overview. Second-Generation Unified GPU Architecture for Visual Computing". Technical Brief (2008).

[102] NVIDIA. "NVIDIA CUDA Compute Unified Device Architecture Programming Guide" (2007).

[103] NVIDIA. "Parallel Thread Execution ISA" (2015).

[104] Netlib Repository. "BLAS (Basic Linear Algebra Subprograms)". http://www.netlib.org/blas/ (2015).

[105] Netlib Repository. "LAPACK - Linear Algebra PACKage". http://www.netlib.org/lapack/ (2015).

[106] NVIDIA. "NVIDIA's Next Generation Compute Architecture: Fermi". Whitepaper (2009).

[107] NVIDIA. "NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110/210". Whitepaper (2014).

[108] NVIDIA. "NVIDIA GF100". Whitepaper (2010).

[109] NVIDIA. "TESLA C2050 and TESLA C2070 Computing Processor Board". Board Specification (2011).

[110] NVIDIA. " CUDA Toolkit 4.0 Feature and Overview Webinar " (2011).

[111] LLVM. "The LLVM Compiler Infrastructure". http://llvm.org (2015).

[112] M. NVIDIA PARALLEL FORALL. "Maxwell: The Most Advanced CUDA GPU Ever Made". http://devblogs.nvidia.com/parallelforall/maxwell-most-advanced-cuda-gpu-ever-made/ (2014).

[113] J.-H. Huang. "Keynote: Leaps in Visual Computing". GPU Technology Conference. NVIDIA (2015).

[114] I. Buck. "NVIDIA's Next-Gen Pascal GPU Architecture to Provide 10X Speedup for Deep Learning Apps". http://blogs.nvidia.com/blog/2015/03/17/pascal (2015).

[115] NVIDIA. "NVIDIA NVLink TM High-Speed Interconnect: Application Performance". Whitepaper (2014).

[116] NVIDIA. "CUDA Toolkit Documentation". `http://docs.nvidia.com/cuda` (2015).

[117] NVIDIA. "Developing A Linux Kernel Module Using RDMA For GPUDirect". Application Guide (2015).

[118] Mellanox. "Mellanox OFED GPUDirect RDMA". `http://www.mellanox.com/page/products_dyn?product_family=116&mtag=gpudirect` (2015).

[119] Qlogic. "QLogic to Optimize InfiniBand Performance on NVIDIA Tesla GPUs Using NVIDIA GPUDirect (News Release)". `http://ir.qlogic.com/phoenix.zhtml?c=85695&p=irol-newsarticle&id=1432496` (2010).

[120] Mellanox. "Accelerating High Performance Computing with GPUDirect RDMA". GPU Technology Conference. NVIDIA (2013).

[121] J. Kraus. "Introduction to CUDA-aware MPI and NVIDIA GPUDirect". GPU Technology Conference. NVIDIA (2013).

[122] GE Intelligent Platforms. "GPUDirect RDMA" (2013).

[123] R. Bittner, E. Ruf, and A. Forin. "Direct GPU/FPGA communication Via PCI express". *Cluster Computing*, 17(2):339–348 (2014).

[124] Kronos Group. "OpenCL - The open standard for parallel programming of heterogeneous systems". `https://www.khronos.org/opencl/` (2015).

[125] EMPhotonics. "CULA tools". `http://www.culatools.com` (2015).

[126] Innovative Computing Laboratory (ICL), University of Tennessee. "MAGMA". `http://icl.cs.utk.edu/magma/` (2015).

[127] Andreas Klöckner. "PyCUDA". `http://mathema.tician.de/software/pycuda/` (2015).

[128] Continuum Analytics. "Numba". `http://continuum.io/open-source/numba` (2015).

[129] D. P. Greenwood. "Bandwidth specification for adaptive optics systems". *J. Opt. Soc. Am.*, 67(3):390–393 (1977).

[130] D. L. Fried. "Time-delay-induced mean-square error in adaptive optics". *J. Opt. Soc. Am. A*, 7(7):1224–1225 (1990).

[131] P. M. Harrington and B. M. Welsh. "Frequency-domain analysis of an adaptive optical system's temporal response". *Optical Engineering*, 33(7):2336–2342 (1994).

[132] E. Fedrigo, R. Donaldson, C. Soenke, et al. "SPARTA: the ESO standard platform for adaptive optics real time applications" (2006).

[133] R. L.Pettazzi, E.Fedrigo. "Impact of Latency and Jitter on the Performance of Adaptive Optics Systems for ELTs". Presentation, ESO (2012).

[134] Europen Southern Observatory. "ESO E-ELT INSTRUMENT: MAORY". Belgian ESO Industry Day (2015).

[135] A. Basden, D. Geng, R. Myers, et al. "Durham adaptive optics real-time controller". *Appl. Opt.*, 49(32):6354–6363 (2010).

[136] A. G. Basden and R. M. Myers. "The Durham adaptive optics real-time controller: capability and Extremely Large Telescope suitability". *Monthly Notices of the Royal Astronomical Society*, 424(2):1483–1494 (2012).

[137] I. Foppiani, E. Diolaiti, A. Baruffolo, et al. "System overview of the Multi conjugated Adaptive Optics RelaY for the E-ELT". In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7736 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, page 2 (2010).

[138] C. Arcidiacono, L. Schreiber, G. Bregoli, et al. "End to end numerical simulations of the MAORY multiconjugate adaptive optics system" (2014).

[139] E. Diolaiti, C. Arcidiacono, G. Bregoli, et al. "Preparing for the phase B of the E-ELT MCAO module project" (2014).

[140] E. Diolaiti. "The MAORY Multi-Conjugate Adaptive Optics module". Presentation (2013).

[141] N. Dipper, A. Basden, U. Bitenc, et al. "Adaptive Optics Real-time Control Systems for the E-ELT". In S. Esposito and L. Fini, editors, *Proceedings of the Third AO4ELT Conference*. INAF - Osservatorio Astrofisico di Arcetri, Firenze (2013).

[142] R. Barrett, M. Berry, T. Chan, et al. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM e-books. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104) (1994).

[143] A. Basden, T. Butterley, R. Myers, et al. "Durham extremely large telescope adaptive optics simulation platform". *Applied optics*, 46(7):1089–1098 (2007).

[144] A. Basden, R. Myers, and T. Butterley. "Considerations for EAGLE from Monte Carlo adaptive optics simulation". *Applied Optics*, 49(31):G1–G8 (2010).

[145] W. H. Press, S. A. Teukolsky, W. T. Vetterling, et al. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA (1992).

[146] J. Kraus. "Increase Performance with GPU Boost and K80 Autoboost" (2014).

[147] G. Herriot, D. Andersen, J. Atwood, et al. "NFIRAOS: first facility AO system for the Thirty Meter Telescope" (2014).

[148] Malcom Smith. "NFIRAOS RTC". `https://indico.obspm.fr/event/5/session/1/contribution/1` (2016).

[149] M. Smith, D. Kerley, G. Herriot, et al. "Benchmarking hardware architecture candidates for the NFIRAOS real-time controller" (2014).